ALIENs and Continuous Time Economies*

Goutham Gopalakrishna †

October 21, 2022

Abstract

I develop a new computational framework called Actively Learned and Informed Equilibrium Nets (ALIENs) to solve continuous time economic models with endogenous state variables and highly non-linear policy functions. I employ neural networks that are trained to solve supervised learning problems that respect the laws governed by the economic system in the form of general parabolic partial differential equations. The economic information is encoded as regularizers that discipline the deep neural network in the learning process. The sub-domain of the high-dimensional state space that carries the most economic information is learned actively in an iterative loop, enforcing the random training points to be sampled from areas that matter the most to ensure convergence. I utilize a state-of-the-art distributed framework to train the network, which speeds up computation time significantly. The method is applied to successfully solve a model of macro-finance that is notoriously difficult to handle using traditional finite difference schemes.

*I thank my advisor Pierre Collin-Dufresne for invaluable guidance. I also thank Markus Brunnermeier for continuous support. This paper benefitted from discussions with Jonathan Payne, Sebastian Merkel, Yavor Kovachev, Marlon Azinovic, and seminar participants at Princeton University, University of Zurich, and European University Institute.

†EPFL and Swiss Finance Institute. Email-address: goutham.gopalakrishna@epfl.ch

1 Introduction

The past decade has seen a surge in macroeconomic and asset pricing models to capture non-linear global dynamics. Models in continuous time offer the ability to capture complex interactions due to their tractability. While the continuous time models characterizing the global dynamics is certainly an improvement to models with linearized solutions, most of the papers resort to smaller state spaces due to computational bottlenecks. The difficulty is amplified when the state variables are endogenous, correlated, and equilibrium quantities exhibit stark non-linearities. For example, D'Avernas and Vandeweyer [2019] show that even in the case of two space dimensions, the standard finite difference method breaks down since it is difficult to preserve the monotonicity of finite difference schemes. Prior literature has addressed this problem by applying an up-winding scheme to the finite difference method, a technique that is borrowed from fluids dynamics, but it is not guaranteed to work in the presence of correlated state variables.[1] Moreover, finite difference methods are not easily scalable, especially when an implicit scheme is employed. This is because an implicit scheme results in a large linear system to be solved, which quickly becomes computationally infeasible when the state space dimension grows.

In this paper, I propose a distributed deep-learning based technique called Actively Learned and Informed Equilibrium Nets (ALIENs), which can be used to solve a large class of continuous time models in financial economics featuring highly non-linear equilibrium policies and endogenous and correlated state variables with heterogeneous agents. The contribution of the paper is two-fold. First, I present a general setup of a heterogeneous agent portfolio choice problem and demonstrate how one can solve the model by approximating the Hamilton-Jacobi-Bellman equation using neural networks. I prove that ALIENs with at least one hidden layer offer theoretical convergence to the resulting PDEs when the number of neurons in the hidden layer is large. This result can be thought of as a universal approximation theorem for quasi-linear parabolic PDEs that are ubiquitous in continuous time finance and macroeconomic models. Second, I solve two models that feature heterogeneous agents and endogenous state variables. The first model is inspired by Brunnermeier and Sannikov [2016]. It serves as a benchmark to test

[1] See Brunnermeier and Sannikov [2016],Achdou et al. [2014b],Gomez [2019] etc., for the application of up-winding scheme.

the neural network approach. I show that the equilibrium policies obtained from a finite difference scheme with up-winding match the solution obtained from ALIENs. The second model adds more shocks to the benchmark model in the spirit of Hansen et al. [2018] and shows how active learning plays a crucial role in ensuring convergence.

The models demonstrated in this paper feature endogenous jumps in return volatilities and risk prices, and are canonical examples of a macro-finance model with financial frictions that have received substantial attention in the literature. In both the benchmark model and the richer model, the wealth share of the intermediary sector is an endogenous state variable. The value function and other equilibrium policies are a function of the wealth share, which moves endogenously with respect to the underlying shocks in the economy. Many of the heterogeneous agent asset pricing models have a similar structure where the model boils down to solving a system of elliptical PDEs. These PDEs are typically highly non-linear and introduce instabilities when solved using standard methods like the finite difference scheme. Even seemingly small errors get propagated over time and the equilibrium policy functions end up with instabilities similar to 'Gibbs phenomenon' in the spectral theory literature, especially when the policies have jumps.[2] To tackle this problem, it is standard in the literature to convert them into a system of quasi-linear parabolic PDEs by adding a pseudo time dimension. This practice, otherwise known as 'false-transient' method, transforms an elliptical PDE into a parabolic PDE and enables a marching solution (Mallinson and de Vahl Davis [1973]). While most of the asset pricing literature in the past have dealt with parabolic PDEs in one or two space dimensions, solving such PDEs in higher dimensions, remain an open challenge. There is a large literature in the applied mathematics area that deals with PDEs in high dimensions but most of them deal with simple problems that do not feature endogenous state variables with highly non-linear functions as PDE coefficients, that in turn endogenously depend on the policy and the value functions.

Recent applications of deep learning in economics aim to directly approximate the equilibrium functions by simulating the ergodic density. For example, Azinovic et al. [2019] solve discrete time overlapping generations problems by training a deep neural network on simulated data, thereby side-stepping the need for labeled points. They argue that in most cases, the ergodic distribution of state variables lies in a small sub-

[2]Gibbs phenomenon refers to the instability at jump discontinuous points when Fourier series is used to approximate a function with jump discontinuity. See Kelly [1996] for a nice exposition.

space of the entire state space, and hence one can solve for the equilibrium functions in this small subspace without spending resources and time on part of the state space that does not matter. Simulating from the ergodic density is computationally cheap if a closed form solution for the distribution exists. However, in the cases where state variables are endogenous and depend on the policy and value functions, these functions have to be approximated first before simulating the state variables through a Monte-Carlo procedure. In continuous time macro-finance models, the transformation of elliptical PDEs into parabolic PDEs already introduces some degree of error. A Monte-Carlo type simulation of state variables using approximated policy and value functions adds further approximation errors. Secondly, typical problems in asset pricing and macro-finance are characterized by high non-linearities in recessionary periods, which are rare occurrences. Therefore, one has to simulate an extraordinarily large number of points in order to have a realistic mass in the state space corresponding to recessions in order to better approximate the equilibrium functions in that subspace and avoid instabilities in future iterations. The framework that I propose takes a different approach. Sparse training points are sampled throughout the state space, but the active learning enforces the sampling to come from the regions with a low tolerance for approximation error from the neural networks. Such a smart-sampling procedure minimizes the error in the initial time iterations so that instabilities in the future iterations are prevented.

I take advantage of recent advancements in high performance computing and offer a parallelized solution using an open-source library developed by Uber called Horovod (see Sergeev and Balso [2018a]). Horovod provides a framework for data parallelism where the input is split into mini-batches and transmitted across several nodes, where each node is accompanied by many GPUs and/or CPUs. The model in each node is the same but receives different inputs for training the neural network. Using a Message Passing Interface (MPI) that enables communication across the nodes, the output from each node is averaged using a ring-Allreduce operation. This procedure significantly speeds up the computation time and is similar in spirit to the parallelization scheme in Azinovic et al. [2019]. The examples that I have chosen in this paper come from macro-finance, but the framework can be applied to a variety of problems in finance. For example, a continuous time version of Bansal and Yaron [2004] with mean-reverting long run growth yields a parabolic PDE for the price-dividend ratio. Other problems in

asset pricing where this framework is applicable include Wachter [2013], Gârleanu and Panageas [2015], Haddad [2012], Di Tella [2017], Gomez [2019], Di Tella [2019] among others. In macroeconomic models where constraints play an important role, ALIENs offer the possibility to easily encode the economic information from the constraints as regularizers. Importantly, if there are multiple constraints, the framework allows to attach different weights to them depending on their importance in the economic problems. Models where constraints play an important role and can be solved using ALIENs are Achdou et al. [2017], Bolton et al. [2011] etc., among others. While most of these papers are restricted to smaller state spaces due to the curse of dimensionality, ALIENs provides scalability that can be leveraged to solve problems in larger dimensions. For example, Gopalakrishna [2020] shows that a two-dimensional macro-finance model can jointly explain various macroeconomic and asset pricing moments.

The paper is organized as follows. Section 2 contains the literature review. Section 3 presents a general setup of portfolio choice problem. Section 4 demonstrates the application of ALIENs to a benchmark model. Section 5 builds a richer model of capital misallocation and solves it using the proposed framework. Section 6 concludes the paper.

2 Literature Review

This paper relates mainly to two strands of literature. First, there has been an explosion of macroeconomic models with financial frictions to characterize the global dynamics after the financial crisis of 2008. The older literature in this area can be traced to Kiyotaki and Moore [1997] and Bernanke et al. [1998] who study the problem of financial acceleration in a discrete time setting. In the past decade, seminal contributions in this area have come from Brunnermeier and Sannikov [2014], He and Krishnamurthy [2013], Di Tella [2017], He and Krishnamurthy [2019], and Adrian and Boyarchenko [2012] using continuous time machinery since setting up problems in continuous time offers some degree of tractability that the discrete time models fail to provide. Other related papers in the asset pricing area such as Basak and Cuoco [1998], Basak and Shapiro [2001], Gromb and Vayanos [2002], Gârleanu and Pedersen [2011] etc. study an endowment economy. Given the potential offered by continuous time framework, a second wave

of papers emerged capturing more complex dynamics of macroeconomic and financial sector. Drechsler et al. [2018], Di Tella [2017], and Silva [2020] analyze the impact of monetary policy, and Adrian and Boyarchenko [2012], Caballero and Simsek [2017] study macro-prudential policies. Often times, continuous time macro-finance models do not admit full fledged closed form solutions and typically involve solving for a system of partial differential equations (PDEs) to obtain the policy functions.

Hansen et al. [2018] and D'Avernas and Vandeweyer [2019] provide robust solutions to solve the PDEs using some variations of finite difference schemes. Hansen et al. [2018] proposes an implicit finite difference scheme with up-winding and employ parallelization techniques to tackle the problem of solving large linear systems. The technique proposed in this paper dominates their approach since it offers the advantage of ease in setting up the numerical scheme in two ways. Firstly, my approach allows to accommodate different types of HJB equations with relative ease. For instance, adding a jump term to the capital process will alter the HJB equation and requires setting up a different linear system to solve in the case of an implicit finite difference scheme. This process can be painful depending on the nature of the problem. On the contrary, adding a jump term in my approach requires simply augmenting the regularizer by adding a derivative term, which can easily be accomplished through automatic differentiation. Secondly, adding more features to the existing model and scaling up the dimension again requires setting up a new linear system to solve in the case of an implicit finite difference scheme, whereas, this can be accomplished by simply adding the required higher order partial derivatives to the regularizers in my approach. Moreover, my approach works for any arbitrary time step, whereas, the method in Hansen et al. [2018] suffers from the problem of using appropriate guess for the time step and space dimension step, as they are tightly linked to guarantee convergence. Experimentation shows that for problems with capital misallocation, the time step that needs to be set is very small which significantly decreases the speed of convergence. D'Avernas and Vandeweyer [2019] explores a similar macro-finance model and demonstrate the difficulty in maintaining the monotonicity of finite difference schemes in solving PDEs, and offer a robust solution technique based on Bonnans et al. [2004]. The technique involves solving for the right direction to approximate the finite differences so as to preserve the monotonicity which is a necessary condition for convergence. However, the approach

that they offer is specific to two space dimensions. On the contrary, ALIENs are easily scalable with minimal effort in coding.

The second related strand of literature is application of machine learning to solve equilibrium models in economics and finance. The papers that are closer are Duarte [2017] and Fernández-Villaverde et al. [2020] since they also consider equilibrium problems in continuous time. Duarte [2017] encodes policy and value functions with neural networks and performs policy evaluation and policy update in the spirit of reinforcement learning. While Duarte [2017] focusses on representative agent asset pricing models that admit analytical solutions, the problems that I consider are heterogeneous agent macro-finance models with endogenous state variables and jumps in policy functions. These problems do not have closed form solutions and are more complex to solve. Fernández-Villaverde et al. [2020] also employs neural networks to solve a model based on Krusell and Smith [1998] in continuous time. However, it is the law of motion of the aggregate wealth that is solved using neural networks. The value function in their model is solved using traditional finite difference method which is in contrast to ALIENs. Azinovic et al. [2019] considers discrete time economic problems and solves for the equilibrium policy functions using neural networks. They also parallelize their algorithm using Horovod to speed up the performance. While their method is based on simulating from the ergodic density, I consider sparse grid padded with active points for training.

There is a substantial literature in computational physics and applied mathematics to approximate PDEs and HJBs using neural network, starting from Sirignano and Spiliopoulos [2018b] and Raissi et al. [2017]. Sirignano and Spiliopoulos [2018b] proposes deep galerkin method to solve PDEs in high dimensions and incorporates monte carlo methods to compute second order derivatives to speed up computation. Raissi et al. [2017] solves canonical two space dimensional problems in computational physics such as Naiver-Stokes and Burgers equations. Han et al. [2018] represents quasi-linear PDEs in the form of forward backward stochastic differential equations and then applies deep neural networks to solve the PDEs. They find that this strategy enables efficient computation of gradients in terms of speed an accuracy. While Raissi et al. [2017], Raissi et al. [2019] etc. use feed-forward neural networks, Sirignano and Spiliopoulos [2018b] finds that advanced architectures like LSTMs offer improved performance. More novel

architectures like convolution networks, which are mostly used in image recognition, have also been found to be useful in solving PDEs (Tompson et al. [2017]). In contrast to these papers, the framework that I propose is suited to solve problems in economic and finance. For instance, many problems in macro-finance and asset pricing come with endogenous state variables that are often correlated- a feature that the models in applied mathematics does not typically deal with. Moreover, the non-linearity of the PDEs in the economic models come from the fact that the advection, diffusion, linear, and cross term coefficients of the PDE are endogenously dependent on the equilibrium policy functions. For example, these coefficients in Brunnermeier and Sannikov [2016] are solved for using a separate Newton-Raphson method since one needs to solve for an algebraic first order differential equation to get these coefficients. These kinds of complications do not arise in the models considered in applied mathematics where more often the coefficients are known constants or simple exogenous functions.[3] Thus, one cannot simply apply the deep learning tools developed in other areas and hope to solve models in financial economics. Lastly, the usage of sparse and active points in this paper relates to the literature on adaptive sparse grids that are concerned with a systematic way of generating the state space grid. For example, Brumm and Scheidegger [2017] uses adaptive sparse grids to solve dynamic economic models, whereas Bungartz et al. [2012] solves option pricing models using finite element method.[4]

3 General Set-up

In this section, I present a general set-up of a portfolio choice problem with a continuum of agents indexed by j, who have a lifetime recursive utility given by

$$U_{j,t} = E_t\Big[\int_t^\infty f(c_{j,s}, U_{j,s})ds\Big] \tag{1}$$

[3]The nature of coefficients play an important role in finite difference methods. For example, see Brunnermeier and Sannikov [2016], who use an up-winding scheme in order to deal with the case where the advection coefficients have different signs in different parts of the state space.

[4]These papers propose a systematic way of refining grid, for example by modifying the basis functions as in Brumm and Scheidegger [2017]. On the contrary, I use a random sample from the mesh that are padded with active points for training the neural network.

with

$$f(c_{j,t}, U_{j,t}) = \frac{1-\gamma}{1-\frac{1}{\varrho}} U_{j,t} \left[\left(\frac{c_{j,t}}{\left((1-\gamma)U_{j,t}\right)^{1/(1-\gamma)}} \right)^{1-\frac{1}{\varrho}} - \rho \right] \tag{2}$$

where ρ, γ, and ϱ are the discount rate, the risk aversion, and the inter-temporal elasticity of substitution (IES) of the agents. I assume that these parameters are the same for all agents in the economy but this is purely for simplicity and can be easily relaxed to introduce further heterogeneity. The agents trade a risky asset, a claim to the dividend denoted by y_t, where $t \in [0, \infty]$, that follows an exogenous process

$$\frac{dy_t}{y_t} = g dt + \sigma dZ_t \tag{3}$$

where Z_t is the standard Brownian motion representing the aggregate uncertainty in $\mathcal{F}_t$, g is the growth rate, and σ is the volatility. The agents also trade in a risk-free security that pays a return r_t that will be determined in the equilibrium. The price of the risky asset q_t is governed by

$$\frac{dq_t}{q_t} = \mu_t dt + \sigma_t^R dZ_t \tag{4}$$

where the drift μ_t and the volatility σ_t^R are endogenous objects to be determined in the equilibrium. The return on the risky asset is given by $dR_t = \frac{a_j(y_t)}{q_t} dt + \frac{dq_t}{q_t}$, where a_j is an agent specific dividend. The net worth of the agents evolve as

$$\frac{dw_{j,t}}{w_{j,t}} = (r_t + \theta_{j,t}\zeta_{j,t} - \hat{c}_{j,t})dt + \theta_{j,t}\sigma_t^R dZ_t \tag{5}$$

where r_t is the risk-free rate, σ_t^R is the volatility of the return on the risky asset, $\theta_{j,t}$ is the portfolio weight on the risky asset, and $\hat{c}_{j,t}$ is the consumption-wealth ratio. The quantity $a_j(y_t)$ is an agent specific function of the dividends,[5] and $\zeta_{j,t}$ is the price of risk, which may differ across agents due to different dividends. The agents maximize the utility (1) subject to the wealth dynamics (5). Shorting of the capital by the agents is disallowed. The HJB equation for the optimization problem can be written as

$$\sup_{\hat{c}_{j,t}\theta_{j,t}} f(c_{j,t}, U_{j,t}) + E_t(dU_{j,t}) = 0 \tag{6}$$

[5]For instance, agents may have a different tax treatment of dividends in which case the net dividend earned will depend on the investor type. In a production economy, some agents may have a lower productivity rate which gives them a lower dividend like in Brunnermeier and Sannikov [2016].

Due to homothetic preferences, the value function takes the form

$$U_{j,t} = \frac{(J_{j,t} w_{j,t})^{1-\gamma}}{1-\gamma} \tag{7}$$

where the stochastic opportunity set $J_{j,t}$ follows the process

$$\frac{dJ_{j,t}}{J_{j,t}} = \mu^J_{j,t} dt + \sigma^J_{j,t} dZ_t \tag{8}$$

The equilibrium objects $(\mu^J_{j,t}, \sigma^J_{j,t})$ need to be solved. Applying Ito's lemma to $J_{j,t}$, the HJB equation can be written as

$$\begin{aligned} \frac{\rho}{1-1/\varrho} = \sup_{\hat{c}_{j,t}, \theta_{j,t}} & \frac{\hat{c}_{j,t}^{1-1/\varrho}}{1-1/\varrho} \rho J_{j,t}^{1/\varrho - 1} + (r_t + \theta_{j,t}\zeta_{j,t} - \hat{c}_{j,t}) \\ & + \mu^J_{j,t} - \frac{\gamma}{2}(\theta^2_{j,t}(\sigma^R_t)^2 + (\sigma^J_{j,t})^2)) - (1-\gamma)\theta_{j,t}\sigma^R_t \sigma^J_{j,t} \end{aligned} \tag{9}$$

The optimal quantities $(\hat{c}_{j,t}, \theta_{j,t})$ can be found by maximizing the HJB equation. It then remains to solve for the function $J_{j,t}$ which depends on the state variables $\boldsymbol{x} \in \Omega$. Assuming that the number of state variables is d, applying Ito's lemma to $J_{j,t}$, and equating the drift terms,[6] we have

$$\mu^J J = \sum_{i=1}^{d} \mu^{x_i}(\boldsymbol{x}) \frac{\partial J}{\partial x_i} + \sum_{i,j=1}^{d} b^{i,j}(\boldsymbol{x}) \frac{\partial^2 J}{\partial x_i \partial x_j} \tag{10}$$

where $\mu^{x_i}(\boldsymbol{x})$ is the drift of the state variable $x_i \in \boldsymbol{x}$ and $b^{i,j}(\boldsymbol{x}) = \frac{1}{2}\sigma^{x_i}(\boldsymbol{x})\sigma^{x_j}(\boldsymbol{x})$ is the scaled product of volatility of the state variables $\{x_i, x_j\} \in \boldsymbol{x}$. The state variables can be endogenous in which case their drift and the volatility may depend on J. Moreover, the PDE (10) is non-linear because the term μ^J depends on J in a highly non-linear fashion. To see that, note that the function μ^J can be obtained from the HJB equation (9) after

[6] I drop the agent and the time index in order to avoid cluttering of notations.

plugging in the optimal $\hat{c}_{j,t}, \theta_{j,t}$. That is, we have

$$\mu^J = \frac{\rho}{1-1/\varrho} - \frac{\hat{c}_{j,t}^{*1-1/\varrho}}{1-1/\varrho}\rho J_{j,t}^{1/\varrho-1} - (r_t + \theta_{j,t}^* \zeta_{j,t} - \hat{c}_{j,t}^*) + \mu_{j,t}^J - \frac{\gamma}{2}(\theta_{j,t}^{*2}(\sigma_t^R)^2 + (\sigma_{j,t}^J)^2)) - (1-\gamma)\theta_{j,t}^* \sigma_t^R \sigma_{j,t}^J \tag{11}$$

where $\hat{c}^*$ and θ^* are the optimal consumption-wealth ratio and portfolio choice respectively. Non-linear PDEs are in general difficult to solve and the literature addresses this issue by converting the equation (10) into a quasi-linear parabolic PDE by introducing an artificial time derivative. That is, the function μ^J is assumed to be a coefficient whose value is computed based on the value of J from the previous time step. This is similar to Brunnermeier and Sannikov [2016], who solve for the equilibrium quantities in the static inner loop given the value function, which then gets updated in the outer time loop. This also means that μ^{x_i} and σ^{x_i} do not depend on J and t but only depend on the equilibrium quantities and the state variables themselves. This two-step method, which is standard in the literature, has relevance to the reinforcement learning where the inner static step is called as 'policy evaluation', and the outer time step is called as 'policy update'. Following the tradition, I add a false time derivative to the PDE (10) and rewrite it in a general quasi-linear parabolic form at the k-th time iteration as

$$\mathcal{G}[J] := \frac{\partial J}{\partial t} + A\left(\boldsymbol{x}, J, \frac{\partial J}{\partial \boldsymbol{x}}\right) + \frac{1}{2} tr\left[B\left(\boldsymbol{x}, \tilde{J}, \frac{\partial \tilde{J}}{\partial \boldsymbol{x}}\right) \frac{\partial^2 J}{\partial \boldsymbol{x}^2} B\left(\boldsymbol{x}, \tilde{J}, \frac{\partial \tilde{J}}{\partial \boldsymbol{x}}\right)^T\right] = 0 \tag{12}$$

$$(t, \boldsymbol{x}) \in [T - k\Delta t, T - (k-1)\Delta t] \times \Omega \tag{13}$$

with the boundary conditions

$$J(t, \boldsymbol{x}) = \tilde{J}; \quad \forall (t, \boldsymbol{x}) \in (T - (k-1)\Delta t) \times \Omega \tag{14}$$

$$\frac{\partial J(t, \boldsymbol{x})}{\partial \boldsymbol{x}} = 0; \quad \forall (t, \boldsymbol{x}) \in (T - (k-1)\Delta t) \times \partial\Omega \tag{15}$$

where in this case, $A\left(\boldsymbol{x}, J, \frac{\partial J}{\partial \boldsymbol{x}}\right) = \sum_{i=1}^{d} \mu^{x_i}(\tilde{J}) \frac{\partial J}{\partial x_i} - \mu^J J$ and $B\left(\boldsymbol{x}, \tilde{J}, \frac{\partial J}{\partial \boldsymbol{x}}\right) = \sigma^{x_i}(\tilde{J})$. The first boundary condition contains $\tilde{J}$ which is the value obtained in previous time iteration. For example, in kth time step, we solve for the function $J(T - k\Delta t, \boldsymbol{x})$ and $\tilde{J}$ in this case denotes the value $J(T - (k-1)\Delta t, \boldsymbol{x})$. The PDE (12) occurs widely in the macro-finance

and asset pricing literature including Brunnermeier and Sannikov [2016], Hansen et al. [2018], Kurlat [2018], Di Tella [2017], Di Tella [2019], Drechsler et al. [2018], Gomez [2019], Krishnamurthy and Li [2020], Li [2020], He and Krishnamurthy [2013], D'Avernas et al. [2019], among others. These papers address different economic problems in varied ways but all of them boil down to solving the PDE (12) subjected to some boundary conditions. In general, a closed-form solution to such PDEs do not exist. The literature has so far used finite difference method with up-winding, which works well in smaller dimensions but becomes infeasible as d grows large due to the curse of dimensionality and the difficulty in preserving the monotonicity of the finite difference scheme.

3.1 Neural network for PDEs

The approximation of function J using a feed-forward deep neural network is done by random sampling of points from the state space making it a mesh-free procedure. This empowers the approach with scalability by alleviating the curse of dimensionality in the PDE time step. Moreover, since the neural network can approximate any type of quasi-linear parabolic PDE, one doesn't need to worry about approximating the derivatives in the right spatial dimensions to preserve monotonicity as in the grid-based finite difference method (D'Avernas and Vandeweyer [2019]).

Feed-forward network: I present a brief introduction to the simple feed-forward neural networks from which the informed neural network is built to solve the PDE (12). A single-layer neural network that can approximate J is given by

$$\hat{\boldsymbol{J}}(\boldsymbol{x}|\Theta) = g\Big(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}\Big) \tag{16}$$

where $\boldsymbol{W}, \boldsymbol{b} \in \Theta$ are the parameters called *weights* and *biases* respectively,[7] and $g(\cdot)$ is the *activation function* which maps the input to the output in a non-linear fashion. The universal approximation theorem (Hornik [1991]) states that any Borel measurable function can be approximated by a feed-forward neural network with a single hidden layer. That is, for any $\epsilon > 0$ and any function $\boldsymbol{J(x)}$ with state variables $\boldsymbol{x} \in I_d$, where I_d is the

[7] *Bias* is a terrible terminology that is unfortunately commonplace in the deep learning literature. We can think of b more as a shift parameter.

d-dimensional unit hypercube, the approximation (16) satisfies[8]

$$|\hat{\boldsymbol{J}}(\boldsymbol{x}|\Theta) - \boldsymbol{J}(\boldsymbol{x})| < \epsilon \qquad \forall \boldsymbol{x} \in I_d$$

The activation functions can be thought of basis functions but they have very simple functional forms as opposed to complex forms such as Chebychev and Legendre polynomials that are commonly used in the projection methods. One minor shortcoming of the universal approximation theorem is that it does not specify the exact number of neurons required to achieve convergence. That is guided entirely by the empirical procedure. It turns out that a single hidden layer network may not provide a good approximation for the function J and hence it requires to stack multiple hidden layers on top of one another to construct a deep neural network. Then, we have

$$\begin{aligned}
\boldsymbol{z}_1 &= \boldsymbol{W_1}\boldsymbol{x} + \boldsymbol{b_1} \\
\boldsymbol{h_1} &= g(\boldsymbol{z}_1) \\
\boldsymbol{z}_2 &= \boldsymbol{W_2}\boldsymbol{h_1} + \boldsymbol{b_2} \\
\boldsymbol{h_2} &= g(\boldsymbol{z}_2) \\
&\vdots \\
\boldsymbol{z}_l &= \boldsymbol{W}_l\boldsymbol{h}_{l-1} + \boldsymbol{b_l} \\
\hat{\boldsymbol{J}} &= \sigma(\boldsymbol{W}_{l+1}\boldsymbol{z}_l + \boldsymbol{b}_{l+1})
\end{aligned}$$

where l is the number of hidden layers and $g(\cdot)$ are the activation functions that remain the same in each hidden layer. The commonly used activation functions in deep-learning literature are Rectified Linear Unit (ReLu), tanh, sigmoid, and shifted-ReLu. I consider tanh activation function for the hidden layers and a sigmoid activation function in the output layer based on superior performance for solving the PDEs. The activation function in hidden layer takes the form

$$g(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{17}$$

[8]This activation function $g(\cdot)$ is not dependent on J and has a natural relation with the Galerkin method if we compare (16) and (34).

The sigmoid function is given by

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{18}$$

Note that the sigmoid function gives values in the range $(0,1)$. This works for the problems considered in this paper since the stochastic opportunity set J in equilibrium is equal to the consumption-wealth ratio, which lies in the range between 0 and 1. If the opportunity set takes values in the range $(-\infty,+\infty)$, then a linear output layer is recommended. The output from the feed-forward deep neural network $\hat{J}(x|\Theta)$ forms the basis for solving the equation (12).

Informed neural nets: The approximation from simple feed-forward network will clearly be poor because it does not encode any information from the PDE. The next logical step is to transform the simple feed-forward network into a more informed network by encoding the economic information into it. I build customized loss-functions that act as regularizers in the neural network optimization. Consider the PDE residual from[9] (12)

$$f := \frac{\partial \hat{J}(\Theta)}{\partial t} + A\left(\boldsymbol{x}, \hat{J}(\Theta), \frac{\partial \hat{J}(\Theta)}{\partial \boldsymbol{x}}\right) + \frac{1}{2} tr\left[B\left(\boldsymbol{x}, \tilde{J}, \frac{\partial \tilde{J}}{\partial \boldsymbol{x}}\right) \frac{\partial^2 \hat{J}(\Theta)}{\partial \boldsymbol{x}^2} B\left(\boldsymbol{x}, \tilde{J}, \frac{\partial \tilde{J}}{\partial \boldsymbol{x}}\right)^T\right] \tag{19}$$

where $\tilde{J}$ denotes the value obtained from the previous time iteration.[10] Starting from a simple feed-forward neural network $\hat{J}(\boldsymbol{x}|\Theta)$ that is parameterized by an arbitrary Θ, the goal is to find the optimal Θ^* that ensures $|\hat{J}(t,\boldsymbol{x}|\Theta) - J| < \epsilon$ for all $\epsilon > 0$. Towards this goal, I minimize the loss function that encodes the economic information from the PDE and the inner static loop. The loss function is given by

$$\mathcal{L} = \lambda_f \mathcal{L}_f + \lambda_j \mathcal{L}_j + \lambda_b \mathcal{L}_b + \lambda_c^1 \mathcal{L}_c^1 + \lambda_c^2 \mathcal{L}_c^2 \tag{20}$$

[9] I denote $\hat{J}(t,\boldsymbol{x} \mid \Theta)$ as $\hat{J}(\Theta)$ for brevity.

[10] Note that the volatility and advection coefficient terms are computed based on the value of J from the previous time iteration. Thus, they are simply considered as coefficients.

where

$$\text{PDE loss} \qquad \mathcal{L}_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(\boldsymbol{x}_f^i, t_f^i)|^2 \tag{21}$$

$$\text{Bounding loss-1} \qquad \mathcal{L}^j = \frac{1}{N_j}\sum_{i=1}^{N_j} |\hat{J}(\boldsymbol{x}_j^i, t_j^i) - \tilde{J}^i|^2 \tag{22}$$

$$\text{Bounding loss-2} \qquad \mathcal{L}^b = \frac{1}{N_b}\sum_{i=1}^{N_b} \left|\frac{\partial \hat{J}}{\partial \boldsymbol{x}_b^i}\right|^2 \tag{23}$$

$$\text{Active loss-1} \qquad \mathcal{L}_c^1 = \frac{1}{N_c}\sum_{i=1}^{N_c} |\hat{J}(\boldsymbol{x}_c^i, t_c^i) - \tilde{J}^i|^2 \tag{24}$$

$$\text{Active loss-2} \qquad \mathcal{L}_c^2 = \frac{1}{N_c}\sum_{i=1}^{N_c} |f(\boldsymbol{x}_c^i, t_c^i)|^2 \tag{25}$$

The points $(\boldsymbol{x}_j^i, \tilde{J}^i)_{i=1}^{N_j}$ and $(\boldsymbol{x}_j^i)_{i=1}^{N_b}$ denote the training sample from the two boundary conditions, $(\boldsymbol{x}_c^i, \tilde{J}^i)_{i=1}^{N_c}$ are the training sample from the active loss region, and $(\boldsymbol{x}_f^i)_{i=1}^{N_f}$, and $(\boldsymbol{x}_c^i)_{i=1}^{N_c}$ correspond to the training samples used to compute the PDE residuals. The loss function is customized to take into account the economic problem at hand. Figure (1) presents the architecture of this network. I prove a convergence theorem, along the lines of Sirignano and Spiliopoulos [2018a], related to the neural network approximation of quasi-linear parabolic PDEs of the form (12). The L^2 loss $\mathcal{L}$ from (20) is a measure of how well the neural network $\hat{J}(t, \boldsymbol{x}|\Theta)$ approximates the function J that solves the PDE (12). The goal is to make this loss close to zero.

Theorem 3.1. *Consider a class of neural networks $\check{C}^n$ with one hidden layer and n number of neurons. Let us denote $\hat{J}^n(t, \boldsymbol{x}|\Theta)$ as the neural network approximation of the function J that solves (12), and let $\mathcal{L}$ be the loss function given in (20). Then, under certain conditions,*

$$\begin{aligned} &\exists \hat{J}^n(t, \boldsymbol{x}|\Theta) \in \check{C}^n \text{such that} \\ &\mathcal{L}(\hat{J}^n(t, \boldsymbol{x}|\Theta)) \to 0 \quad as \quad n \to \infty \end{aligned} \tag{26}$$

Proof: See Appendix.

That is, as the number of neurons in the hidden layer goes to infinity, the loss converges to zero making the neural network approximation accurate. Note that we only have control of the L^2 error and thus the approximation is accurate in a mean squared error

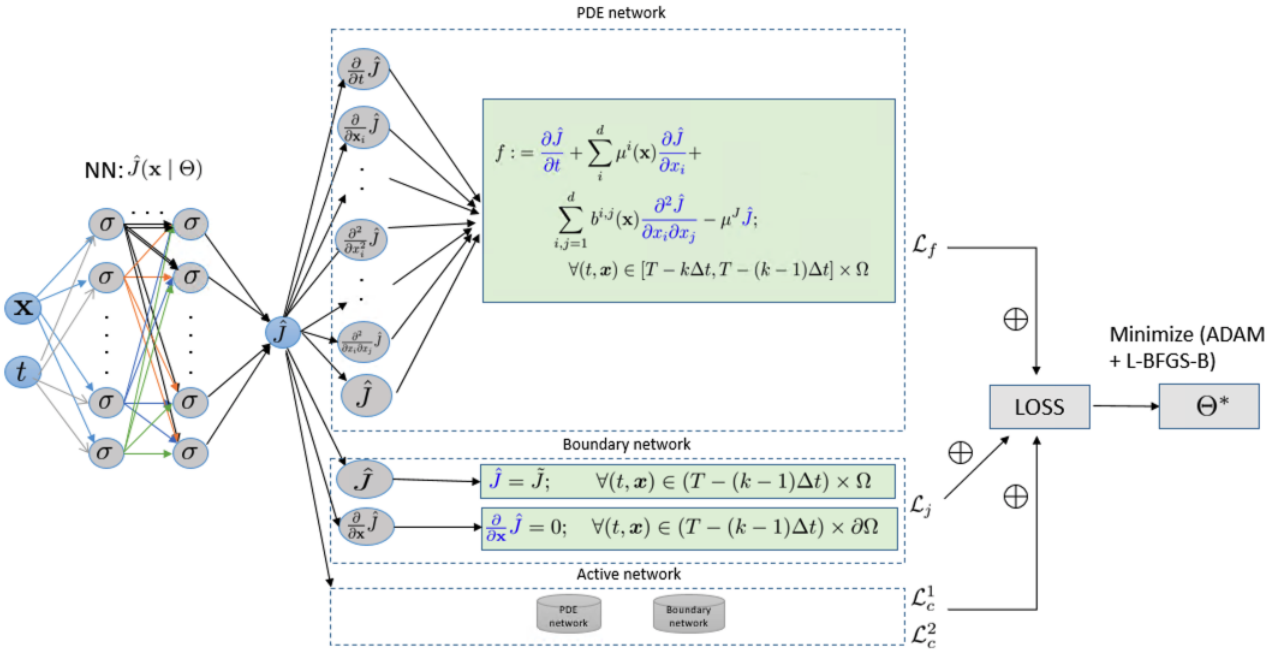

Figure 1. Network architecture

sense.

3.1.1 Under the hood:

The training samples in this method come entirely from the boundary conditions. Unlike the traditional machine learning paradigm where the model is trained on the training set, cross-validated on the validation test, and tested for accuracy on the test set, the method used in this paper does not really have training samples in the same sense. The inputs from boundary conditions can be thought of as pseudo-training samples that are used to approximate the function J. As a result, the problem of overfitting, which is ubiquitous in the machine learning paradigm is less relevant here. Moreover, only a fraction of data points are sampled in each PDE step iteration and hence any minor concerns of overfitting is eliminated.[11] The feed-forward architecture shown in Figure (1) is simple compared to the more complex ones such as GANs, Autoencoders, and LSTMs used in the asset pricing literature (see Gu et al. [2020], Chen et al. [2019] etc). This raises the question what makes the algorithm succeed in learning the function J in a high dimensional space. The answer lies in the encoding of economic information through customized loss functions that makes the simple feed-forward network more informed. The PDE loss function dictates the neural networks to satisfy the HJB equa-

[11] This is similar to using mini-batches in deep learning to reduce overfitting.

tion such that the residual from the HJB is close to zero in a mean-squared sense. At the same time, the neural networks are also forced to obey the initial/boundary conditions through the bounding loss functions. The goal is then to optimize for the parameters Θ such that the neural network approximation $\hat{J}(\boldsymbol{x}|\Theta)$ minimizes the total loss function. In this optimization problem, the customized loss functions act as *regularizers*.

Automatic differentiation: The success of deep learning is largely due to automatic differentiation, which is a computationally efficient way to compute derivatives of potentially non-linear functions. The neural network approximator works by receiving the input $\boldsymbol{x}$ and computing the output $\hat{J}(\boldsymbol{x}|\Theta)$ that is a composition of simple functions. The derivative of $\hat{J}$ with respect to the inputs $\boldsymbol{x}$ can be obtained analytically by repeated applications of the chain rule. The backpropogation algorithm traverses the graph, computes the derivatives of symbolic variables and stores these operations into new nodes in the graph for later use. To compute a higher order derivative, one can simply run the backpropogation again through the extended graph and obtain it easily. While the deep learning literature uses the automatic differentiation in computing the derivatives of loss function with respect to the parameters such as weights and biases, I use it explicitly to take derivatives with respect to the space and time dimensions. This is illustrated in the PDE network in Figure (1). The derivatives of function $\hat{J}$ with respect to each space and time dimensions are stored as separate nodes in the graph, which are used to compute higher order derivatives through backpropagation. Formally, the cost of computing $\dfrac{\partial J}{\partial \boldsymbol{x}}$ is $O(d) \times \text{cost}(J)$, which is the same as the cost of computing $\dfrac{\partial^2 J}{\partial^2 \boldsymbol{x}}$ since the backpropagation takes advantage of the first order derivatives stored in the computational graph when applying the chain rule. Thus, the explicit use of automatic differentiation to compute derivatives with respect to the space dimension allows fast computation even in high dimensions. On the contrary, computing such higher order derivatives bears a cost $O(d^2) \times \text{cost}(J)$ in finite-difference methods, imposing a bottleneck when d is potentially large. Lastly, the separation of the fundamental neural network and the more informed PDE and bounding network allows us to witness the automatic differentiation fully at action, which is the main driver of the learning process.

Optimization: The total loss function is the weighted sum of the loss from PDE, boundary, and crisis network. I use a combination of adaptive momentum (ADAM) and Limited memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS-B) optimizers to solve for Θ^*. While the fundamental network has only four hidden layers, the PDE network adds more layers since it involves gradient computations with respect to the state variables. Hence, the overall network is quite deep, with a highly complex and non-convex loss function. While ADAM optimizer, which is the standard algorithm in deep learning models, is based on first order derivative, L-BFGS-B is a second order method and is empirically found to be effective in solving the problems considered in this paper. The implementation details are relegated to Appendix A.3.7. The generic algorithm of ALIENs is given in the pseudo-code (1).

Algorithm 1

procedure ALIENS
  Initialize function J, set tolerance level tol
  while $error > tol$ **do**
    Compute policy functions taking as given J
    Construct a neural network approximation $\hat{J}(\Theta)$
    Construct boundary training points
    Construct active training points
    Compute aggregated loss $\mathcal{L}$
    for epoch=1 to maxEpochs **do**
      Create a mini-batch from training points ▷ Optional
      Minimize loss using ADAM optimizer, ▷ Learning rate = 0.001
    Minimize loss using L-BFGS-F until convergence, update Θ^*
    Compute $error := \max|\hat{J}(\Theta^*) - J|$
    Update $J \leftarrow \hat{J}(\Theta^*)$
    Stop if $error < tol$

Active learning: When the equilibrium policy functions have stark non-linearities, a smart sampling procedure is required to ensure faster convergence. In the case of a regime changing policy function, a subdomain in the state space captures the region where the regime shifts. In such cases, it is crucial to have a better approximation of equilibrium functions in this subdomain so as to avoid instabilities in the future time iterations. If the modeler possesses a prior knowledge of the location of this subdomain, then one can obtain a better approximation by sampling sufficiently large number of training points from this subdomain. However, this information is often not known

prior and only gets revealed after the equilibrium policy functions are solved in the first time iteration. Moreover, this subdomain may change from one iteration to next as the policies are evaluated and updated. To tackle this problem, I endow the neural network with an active learning which tracks this subdomain in each iteration by inspecting the equilibrium policy functions. The neural network takes advantage of the fact that the static step precedes the outer time step in an iterative fashion, and actively seeks information about the subdomain at every iteration. Once this subdomain is revealed, active training points are created through random sampling of grid points from this subdomain to construct the loss functions $\mathcal{L}_c^1$ and $\mathcal{L}_c^2$ in (20). The practice of seeking new data points to label in supervised learning is called as *active machine learning*, a budding area in the artificial intelligence literature where the model seeks out new inputs to improve the learning in subsequent iterations. In the context of reinforcement learning, the model learns about new state space by actively interacting with the environment. In the models considered in this paper, the state space is well defined but new grid points are sampled to be used for training the neural network. In doing so, the regularizers $\mathcal{L}_c^1$ and $\mathcal{L}_c^2$ forces the model to learn better in the subdomain since it is costly to have errors in regions of regime shift as they get propagated and amplified in the subsequent iterations. The proposed active learning algorithm alleviates these problems and ensures convergence. The relative weights $(\lambda_f, \lambda_j, \lambda_b, \lambda_c^1, \lambda_c^2)$ control for the importance that each of the components carries in the aggregated loss function.

Let us take a concrete example to understand how active learning works. Consider a two space-dimensional model that generates a capital price as shown in Figure (2). For now, we can abstract away from the details of the model which will be explained in Section 5 and focus on how active learning is applied. The two state variables are the wealth share (z) and the productivity (a) of expert sector in the economy. The model features non-linear equilibrium policy where the capital price is high and almost linear when the wealth share of experts is high, and falls as the wealth share drops. The full state space grid shown in Fig (2) contains tightly packed points in the wealth share dimension since it is the primary driver of price dynamics in the model. The panels (c) and (d) displays the sparse grid used in training the neural networks. The blue points span throughout the state space while the green points span the neighborhood of sharp transition. One can notice that this neighborhood changes from iteration 1 in panel (c)

to iteration 15 in panel (d). As non-linearities are slowly introduced into the parabolic PDEs, the dynamics of the equilibrium prices change and the non-linearities in policy functions occur in different parts of the state space. Not only that, the shape of the sub-domain may also change, as it does in the example demonstrated. By actively tracking the subdomain and sampling points from the neighbourhood of this subdomain, the sparse training sample is guaranteed to have points from the state space that matter the most to avoid instabilities in future iterations. While it it not costly to use the entire grid when the dimension is small, such a smart sampling method is required in higher dimensions where it is infeasible to use the entire grid to train the neural network.

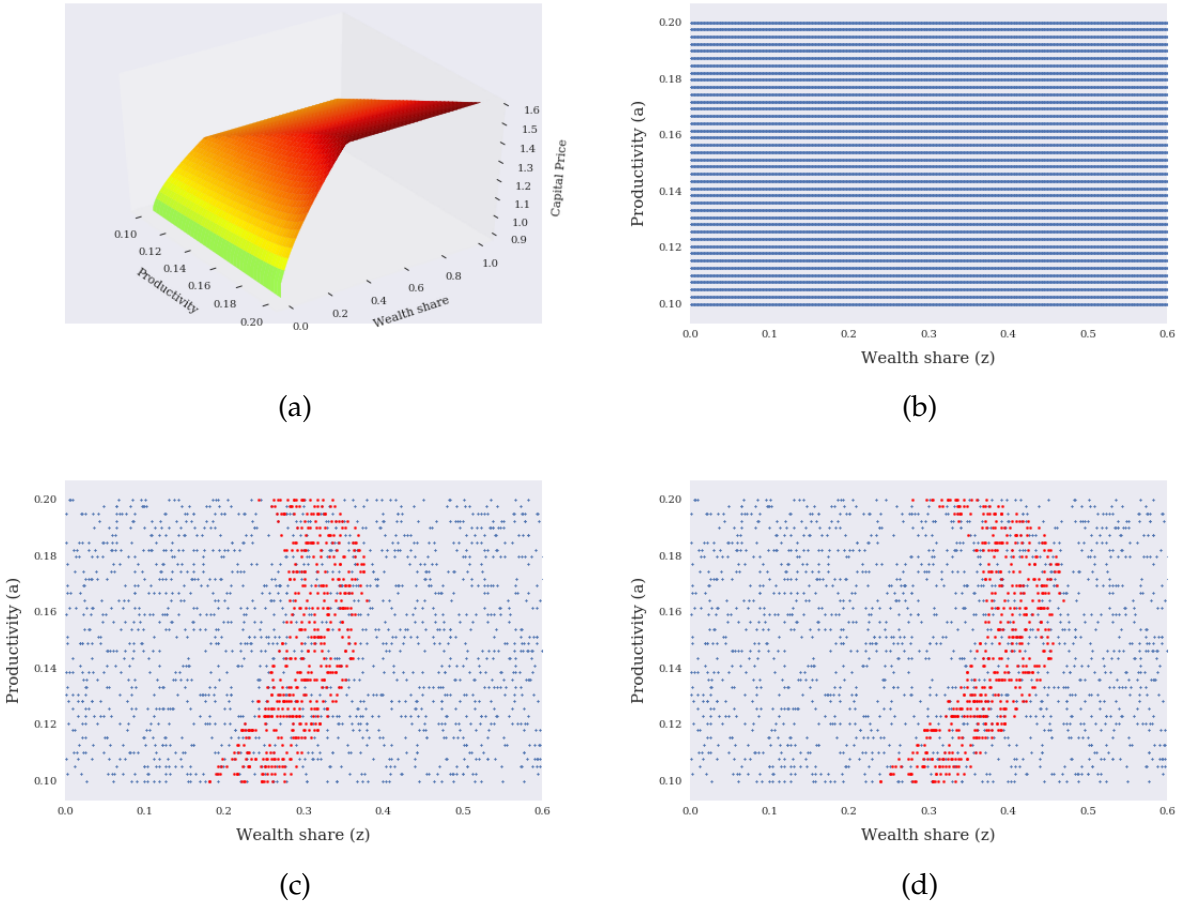

Figure 2. Panel (a) shows Equilibrium capital price. Panel (b) shows full grid with 1000 points in wealth share and 40 points productivity dimension respectively. Panels (c) and (d) shows sparse grid used for learning where points in green represent neighborhood of regime change in the 1^{st} and 15^{th} iteration respectively.

Distributed learning: The deep learning model in this paper is much simpler compared to the state-of-the-art models such as GPT-3 model[12]. Having a lighter model allows us to utilize data parallelism instead of model parallelism, where in the former case, the chief worker[13] in the cluster splits the data and distributes it to several workers that share the same model. Once training is completed, the result is averaged across workers and sent to the chief worker that updates the model with the optimal parameters. In cases where the model is too large to store in a single worker, model parallelism is applicable where each worker trains a piece of the model but with the same data. One challenge in data parallelism is that since each worker shares the same model, the parameters need to be synchronized. I utilize Horovod[14], a popular distributed machine learning framework that manages the internal working of gradient aggregation across the workers through *ring-AllReduce* operation. In some sense, Horovod works like a wrapper around the Message Passing Interface that enables communication across different workers. While data parallelism employed in this paper speeds up computation significantly, it also lends itself nicely to another advantage. As the academic profession moves more towards an open source style research, the data parallelism allows the users to employ big data on models developed by other users. Often, quantitative analysis of economic models requires extensive simulation studies to dissect and understand the underlying mechanisms of the model.[15] Appendix A.3.7 provides the details of Horovod along with sample code to demonstrate the simplicity in usage and deployment.

4 Benchmark Model

In this section, I consider a one space-dimensional model based on Brunnermeier and Sannikov [2016] augmented with recursive preferences that will serve as the benchmark. I solve this model using the proposed neural network method with active learning and show that the solution matches the numerical solution obtained from an up-winding finite difference scheme. Since the closed-form solution to the benchmark model is not

[12]This is a language model by Brown et al. [2020] from OpenAI that recently gained substantial attention. The model has a total of 175 billion parameters to train.

[13]A worker refers to a node in the cluster that is made up of a number of computer nodes.

[14]See Sergeev and Balso [2018a] for details.

[15]See, for example, Gopalakrishna [2020] who uses simulation studies to perform dissection of a macro-finance model.

available, the comparison of the neural network solution is made against the finite difference solution.

4.1 Model

I consider a heterogeneous agent economy populated by households (h) and experts (e) who form a set $\mathbb{H}$ and $\mathbb{E}$ respectively. The aggregate capital in the economy is denoted by K_t where $t \in [0, \infty)$ denotes time. Due to homogeneity of preferences, which will be explained later, we can consider a representative household h and expert e for the rest of the model. Both the households and the experts are allowed to hold capital with a no shorting constraint but the households obtain a lower productivity rate (a_h) compared to the experts (a_e) from investing in the capital. The constraint in the model is such that experts have to retain at least a fraction of the equity in their balance sheet. This is the skin-in-the game constraint that precludes the economy from achieving perfect risk-sharing. The production technology is given by

$$y_{j,t} = a_j k_{j,t}, \qquad j \in \{e, h\}$$

where $k_{j,t}$ is the capital held by agent j whose process is governed by

$$\frac{dk_{j,t}}{k_{j,t}} = (\Phi(\iota_{j,t}) - \delta)dt + \sigma dZ_t^k \tag{27}$$

where $\iota_{j,t}$ is the investment rate, δ is the depreciation rate of capital, σ is the volatility of the capital, and $\{Z_t^k \in \mathbb{R}; \mathcal{F}_t, \Omega\}$ are the standard Brownian motions representing the aggregate uncertainty in $(\Omega, \mathbb{P}, \mathcal{F})$. The quantity $\Phi(\cdot)$ is the investment cost function that is concave and has decreasing returns to scale. The aggregate capital in the economy is denoted by K_t. That is, $K_t = \int_{\mathbb{E}} k_{j,t} dj + \int_{\mathbb{H}} k_{j,t} dj$.

Preferences: Each agent has a continuous-time Duffie-Epstein utility given by

$$U_{j,t} = E_t \left[\int_t^\infty f(c_{j,s}, U_{j,s}) ds \right] \tag{28}$$

where the aggregator takes the form

$$f(c_{j,s}, U_{j,s}) = (1-\gamma)\rho U_{j,t}\left(\log(c_{j,t}) - \frac{1}{1-\gamma}\log\Big((1-\gamma)U_{j,t}\Big)\right) \tag{29}$$

The parameter γ captures the risk aversion of agent j, and the inter-temporal elasticity of substitution (IES) is assumed to be 1. The aggregator (29) is a limiting case of (2).[16] I relegate rest of the model and equilibrium computation to Appendix A.2.1 and go straight to the solution technique. Given the homotheticity of preferences, we can conjecture that the value function is of the form

$$U_{j,t} = \frac{(J_{j,t}K_t)^{1-\gamma}}{1-\gamma} \tag{30}$$

where K_t is the aggregate capital, and the stochastic opportunity set $J_{j,t}$ follows the SDE

$$\frac{dJ_{j,t}}{J_{j,t}} = \mu^J_{j,t}dt + \sigma^J_{j,t}dZ^k_t \tag{31}$$

where the equilibrium objects $\mu^J_{j,t}$ and $\sigma^J_{j,t}$ will have to be determined. The endogenous state variable of the model is the wealth share of experts defined by

$$z_t = \frac{W_{e,t}}{q_t K_t}$$

where $W_{e,t}$ is the aggregated wealth of all experts in the economy. For convenience, the utility (30) is written as a function of capital K_t but they could also be written as a function of $W_{j,t}$. By applying Ito's lemma to $J_{j,t}(z_t)$ and equating the drift terms, we have

$$\mu_{j,t}J_{j,t} = \frac{dJ_{j,t}}{dz_t}\mu^z_t + \frac{1}{2}\frac{d^2J_{j,t}}{dz_t^2}(\sigma^z_t)^2 \tag{32}$$

where the function $\mu^J_{j,t}$ can be obtained from the HJB equation similar to the general setup in Section 3. The solution method involves solving for the equilibrium quantities $(\chi_t, k_{j,t}, q_t, \sigma^q_t)$ using a Newton-Rhaphson method, similar to Brunnermeier and Sannikov [2016], and then updating the value function $J_{j,t}$ which requires solving (32). This equation, when augmented with an artificial time derivative, resembles the quasi-linear

[16] This is for simplicity and can be easily relaxed to the case of non-unitary IES.

parabolic PDE introduced in (12). I relegate the details of the Newton-Rhaphson method used to solve for equilibrium quantities to the Appendix and present the methodology to solving (32) in the main text.

4.2 Traditional methods

The literature has used finite difference method extensively (see Brunnermeier and Sannikov [2016], Di Tella [2017], Di Tella [2019], Gomez [2019], Hansen et al. [2018], D'Avernas and Vandeweyer [2019] etc.) to solve (32). The method works by approximating the derivatives in PDE by discretizing the state space and then solving the discretized problem using an explicit or implicit method with up-winding scheme. I briefly discuss the methodology before presenting the solution. Consider a discretized version of the PDE in (12)

$$J_{i,j+1} = J_{i,j} + \Delta_j \left\{ A\left(x, J_{i,j+1}, \frac{\hat{\partial} J_{i,j+1}}{\hat{\partial} x}\right) + \frac{1}{2} tr\left[B\left(x, J_{i,j}, \frac{\partial J_{i,j}}{\partial x}\right) \frac{\hat{\partial}^2 J_{i,j+1}}{\hat{\partial} x^2} B\left(x, J_{i,j}, \frac{\partial J_{i,j}}{\partial x}\right)' \right] \right\}$$

where $\{i\}_1^{N_z}, \{j\}_1^{N_t}$ denote the space and time dimension grid points respectively, and the derivatives of J are approximated using

$$\begin{aligned}
\frac{\hat{\partial} J_{i,j}}{\hat{\partial} z} &= (\mu_j^x)^+ \frac{J_{i+1,j} - J_{i,j}}{\Delta_i} + (\mu_j^x)^- \frac{J_{i,j} - J_{i-1,j}}{\Delta_i} \\
\frac{\hat{\partial}^2 J_{i,j}}{\hat{\partial} z^2} &= \frac{J_{i+1,j} - 2J_{i,j} + J_{i-1,j}}{\Delta_i^2} \\
\frac{\hat{\partial} J_{i,j}}{\hat{\partial} t} &= \frac{J_{i,j+1} - J_{i,j}}{\Delta_j}
\end{aligned}$$

$$\text{where} \qquad (\mu_j^x)^+ = \begin{cases} \mu_{t,j}^x & \text{if} \quad \mu_{t,j}^x > 0 \\ 0 & \text{if} \quad \text{otherwise} \end{cases} \qquad (\mu_j^x)^- = \begin{cases} \mu_{t,j}^x & \text{if} \quad \mu_{t,j}^x < 0 \\ 0 & \text{if} \quad \text{otherwise} \end{cases} \tag{33}$$

The first derivative of J with respect to space dimension is approximated using a forward difference if the advection coefficient is positive, and using a backward difference if the coefficient is negative, as shown in (33). This ensures monotonicity of the finite difference scheme. The method is implicit since $J_{i,j+1}$ is on both the L.H.S and R.H.S of the discretized PDE equation. This presents a system of linear equations that can be solved using traditional methods such as Richardson method. There are two issues that

arise when applying a finite difference method in high dimensions. The first is the well-known *curse of dimensionality*. With d space dimensions and 1 time dimension, the mesh size in finite difference method is O^{d+1} which becomes infeasible to handle if d is large. Moreover, along with the explosion of the mesh size, there arises a need for reduced time step size. The second problem that has received relatively less attention in the literature is the need to preserve monotonicity of the elliptical operator when the state space is large and potentially correlated. In the case of a one-dimensional state space, the monotonicity can be preserved using an up-winding scheme given in (33) which approximates the derivatives in the right spatial direction. However, as documented in D'Avernas and Vandeweyer [2019], it is not trivial to preserve monotonicity using an up-winding scheme even in a two space dimensional case since the right direction in which derivatives should be approximated may fall outside the grid.

The second method that is commonly used is a projection method (Judd [1992]) where the function J in (12) is approximated using Chebychev polynomials as basis functions. A related technique used extensively in computational engineering is the Galerkin method that finds a linear combination of basis functions that best approximates the PDE. To be precise, suppose the goal is to solve for the PDE $\mathcal{H}(J) = 0$. The approximation for J can be obtained from

$$\tilde{J} = \sum_{i=1}^{n} \alpha_i \phi_i \tag{34}$$

where $\{\phi_i\}$ is bases for J. The residual after plugging in the approximation is denoted as

$$\mathcal{R}(\cdot \mid \boldsymbol{\alpha}) = \mathcal{H}(\tilde{J})$$

The coefficients $\boldsymbol{\alpha}$ satisfy the inner product for $j = 1, ..., n$

$$\left\langle \mathcal{R}(\cdot \mid \boldsymbol{\alpha}), \phi_i \right\rangle = 0$$

One has to choose the basis functions carefully since the computation of inner product can be computationally expensive.

4.3 Model Solution

I calibrate the benchmark model with parameters in Table (1). The goal is to demonstrate that neural network approximation is close to the finite difference approximation, and hence I mostly choose the parameters based on prior literature. The discount rate and depreciation rate of capital is taken from Hansen et al. [2018]. The volatility is chosen to be 6% so as to achieve a reasonable variation in the risk prices. Risk aversion parameter for the agents is set to 2, and the IES is set to 1. Productivity parameters, and the equity retention rate are close to Brunnermeier and Sannikov [2016]. The network architecture and hyperparameters used to solve the de-coupled system of PDEs (32) are provided in the Table (2). I use four hidden layers with 30 neurons in each layer. In principle, one could use one hidden layer and more neurons but experimentation shows that it is better to have a deep network instead of a wide network.[17] The total number of points in the inner static step is 1,000 and the total training sample size to solve the PDE is 300. Figure (4) present the solution obtained using the finite difference method and the neural network method. They not only look qualitatively the same, but they also match upto the order 1e-3. Figure (3) shows the absolute error in value function over time. A comparison with finite difference scheme shows that the neural network method has a larger error drop in each iteration leading to convergence in 20 time steps. Note that the neural network error curve is not as smooth as the finite difference one due to the random sampling from the state space before every PDE time step. However, convergence is achieved in 20 iterations while it takes around 70 iterations for the finite difference method.[18]

[17] It is a common practice in the deep learning literature to have deeper layers instead of wider layers since they have superior performance for a wide range of problems.

[18] A comparison of speed is not appropriate for this benchmark model since neural network takes a long time to train. In smaller dimensions, deep learning suffers from the 'curse of training'. However, in higher dimensions, the computational cost for the deep learning method grows moderately, as opposed to an exponential growth for the finite difference method.

	Description	Symbol	Value
	Volatility of output	σ	0.06
	Discount rate (experts)	ρ	0.05
	Depreciation rate of capital	δ	0.05
Technology/Preferences	Investment cost	κ	10
	Productivity (experts)	a_e	0.15
	Productivity (households)	a_h	0.03
Utility	Risk aversion	γ	2
Friction	Equity retention	$\underline{\chi}$	0.5

Table 1: Calibrated parameters for the benchmark model. All values are annualized.

	Value
No. of hidden layers	4
Hidden units	[30,30,30,30]
Activation function	Tanh (hidden), Sigmoid (output)
Optimizer	ADAM + L-BFGS-B
Learning rate	0.001
Loss function weights($\lambda_f, \lambda_j, \lambda_b, \lambda_c^1, \lambda_c^2$)	{1,1,0.001,1,1}
Batch size	Full batch

Table 2: Network hyperparameters and architecture.

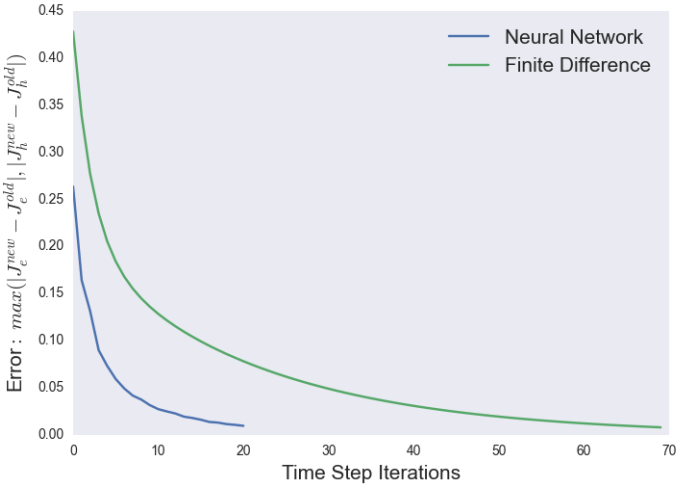

Figure 3. Comparison of value function error in finite difference and neural network method.

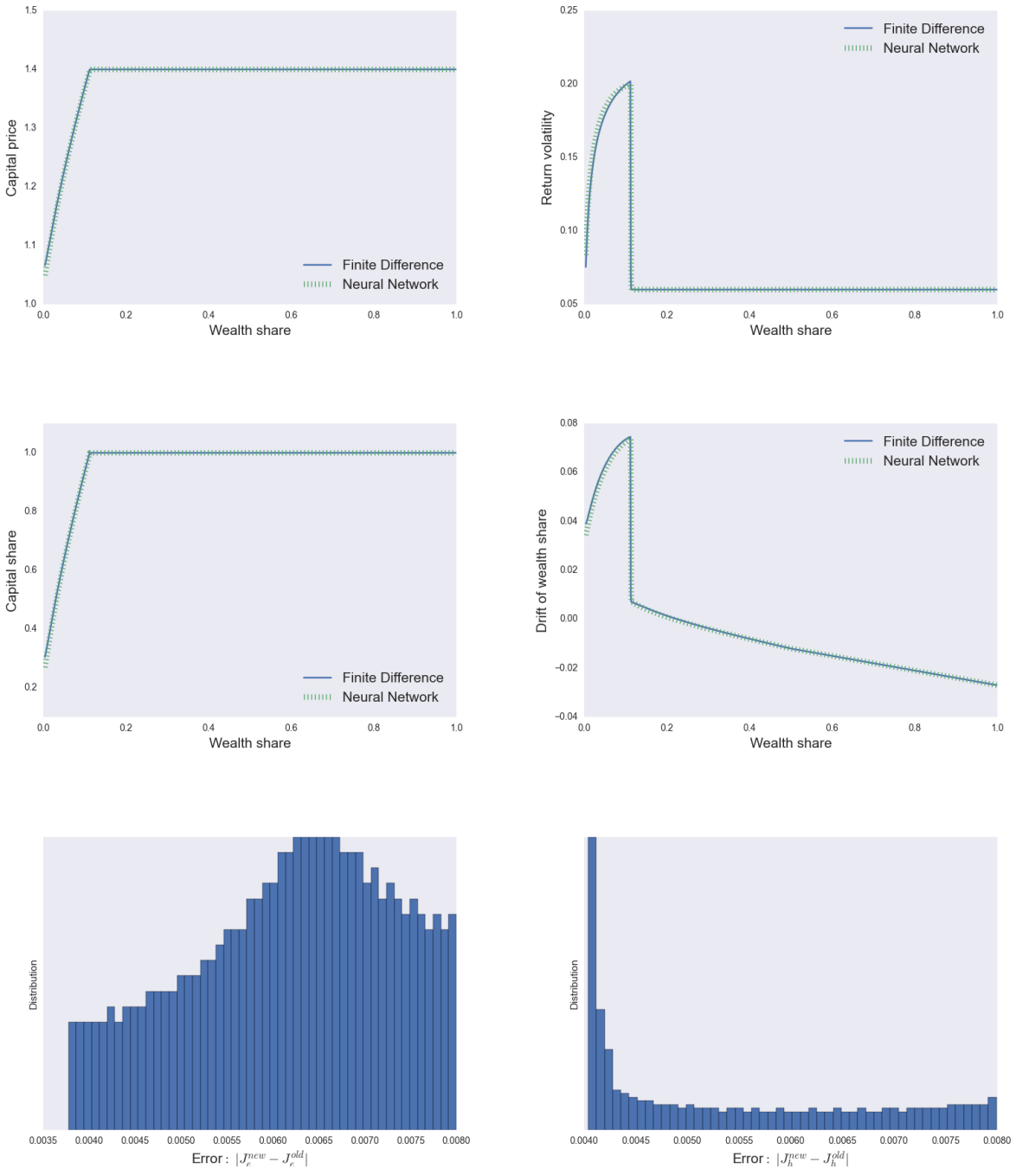

Figure 4. Comparison of equilibrium quantities using finite difference and neural network in the benchmark model. Last two plots correspond to absolute error in functions $\hat{J}_h$ and $\hat{J}_e$.

5 Brunnermeier-Sannikov meets Bansal-Yaron

This section considers a heterogeneous agent model similar to the benchmark case but with more complexity. Specifically, I combine the model of Brunnermeier and Sannikov [2016] and Bansal and Yaron [2004] in continuous time. The economy features heterogeneous agents with experts facing shocks to their productivity and the capital. In addition, the long run growth rate and volatility of capital are subjected to independent Brownian shocks. The state space becomes four dimensional and the PDEs to be solved in the outer time step has five dimensions including the artificial time dimension. I explain the model first and then discuss the results.

5.1 Model

There are two types of agents and the aggregate capital is denoted by K_t with $t \in [0, \infty)$. The capital process is governed by

$$\frac{dk_{j,t}}{k_t} = (\Phi(\iota_{j,t}) + g_t - \delta)dt + \sigma\sqrt{s_t}dZ_t^k \tag{35}$$

where $\iota_{j,t}$ is the investment rate, $\Phi(\cdot)$ is the investment cost function, g_t is the time varying growth rate of capital, and $\sigma\sqrt{s_t}$ is the time varying volatility of capital. The production technology is given by

$$y_{j,t} = a_{j,t}k_{j,t} \tag{36}$$

where $k_{j,t}$ is the capital held by agent type j. I assume that the productivity of experts is time varying and is governed by the process

$$da_{e,t} = \lambda^a(\hat{a}_e - a_{e,t})dt + \sigma^a(a_{e,t})dZ_t^a \tag{37}$$

where λ^a is the mean reversion rate. For simplicity, I assume that the productivity of households is constant $a_{h,t} = a_h$. The growth rate and volatility of capital follows the

exogenous processes

$$dg_t = \lambda^g(\hat{g} - g_t)dt + \sigma^g(g_t)dZ_t^g \tag{38}$$

$$ds_t = \lambda^s(\hat{s} - s_t)dt + \sigma^s(s_t)dZ_t^s \tag{39}$$

where λ^g, λ^s are the mean reversion rates of growth rate and capital processes respectively. I assume that the functions $(\sigma^i(\cdot); i \in \{g, s, a\})$ take the form

$$\sigma^g(\cdot) = (\bar{g} - g_t)(g_t - \underline{g}) \tag{40}$$

$$\sigma^s(\cdot) = (\bar{s} - s_t)(s_t - \underline{s}) \tag{41}$$

$$\sigma^a(\cdot) = (\bar{a}_e - a_{e,t})(a_{e,t} - \underline{a}_e) \tag{42}$$

The assumed functional form above is for convenience in computation and can be modified to resemble a Feller square-root process or an Ornstein-Uhlenbeck process. I impose $\bar{a}_e > \underline{a}_e > a_h$ so that experts productivity is always higher than that of households even thought it is time varying. The Brownian shocks $\{dZ_t^k, dZ_t^g, dZ_t^s, dZ_t^a\}$ have zero cross-correlation for simplicity. Overall, the model can be thought of as a combination of Brunnermeier and Sannikov [2016] and Bansal and Yaron [2004] with additional productivity shocks. The agents trade the capital which has a price process given by

$$\frac{dq_t}{q_t} = \mu_t^q dt + (\boldsymbol{\sigma_t^q})^T \boldsymbol{dZ_t} \tag{43}$$

where the vectors $(\boldsymbol{\sigma^q})^T = \begin{bmatrix} \sigma^{q,k} & \sigma^{q,g} & \sigma^{q,s} & \sigma^{q,a} \end{bmatrix}$ and $\boldsymbol{dZ_t} = \begin{bmatrix} dZ_t^k & dZ_t^g & dZ_t^s & dZ_t^s \end{bmatrix}^T$. The agents can also trade in the risk free security that pays a return r_t. The agents cannot write contracts on the aggregate state of the economy. That is, the investment in capital and the exposure to aggregate shocks $\boldsymbol{Z_t}$ are intertwined. This is for simplicity and can be extended to including a derivative market to hedge the aggregate shocks. This will have an impact on the equilibrium policies and prices but the numerical method to solve the HJB equations will remain the same. Thus, I intertwine the capital holding decisions and the exposure to the aggregate shocks to ease computation of equilibrium policies in the static loop. Since the dividend yield from each unit of the capital held is different

for the households and the experts[19], the agent-specific return process follows[20]

$$dR_{j,t} = \left(\frac{a_{j,t} - \iota_{j,t}}{q_t} + \mu_t^q + \Phi(\iota) - \delta + \sigma\sqrt{s_t}\sigma_t^{q,k} + g_t \right) dt + (\boldsymbol{\sigma_t^R})^T \boldsymbol{dZ_t} \tag{44}$$

where the vector $(\boldsymbol{\sigma_t^R})^T = \begin{bmatrix} \sigma_t^{q,k} + \sigma\sqrt{s_t} & \sigma_t^{q,g} & \sigma_t^{q,s} & \sigma_t^{q,a} \end{bmatrix}$, and $\boldsymbol{dZ_t}$ is as before. The aggregate output in the economy is $y_t = A_t K_t$ where the aggregate dividend is given by

$$A_t = \int_{\mathbb{H}} a_h \frac{k_{j,t}}{K_t} dj + \int_{\mathbb{E}} a_{e,t} \frac{k_{j,t}}{K_t} dj$$

where $K_t = \int_{\mathbb{H}\cup\mathbb{E}} k_{j,t} dj$ denotes the aggregate capital. Let the share of aggregate capital held by the experts be denoted by ψ_t. That is,

$$\psi_t := \frac{\int_{\mathbb{E}} k_{j,t} dj}{\int_{\mathbb{H}\cup\mathbb{E}} k_{j,t} dj}$$

The experts are constrained in issuing equity to the households such that they have to retain a fraction $\underline{\chi} \in [0,1]$ of the equity in their balance sheet. They are free to issue the remaining equity to the households who may desire to hold it in equilibrium. The stochastic discount factor (SDF) process for each type of agent is given by

$$\frac{\partial \xi_{j,t}}{\xi_{j,t}} = r_t dt - \boldsymbol{\zeta}_{j,t}^T \boldsymbol{dZ_t} \tag{45}$$

where the vector $\boldsymbol{\zeta}_{j,t}^T = \begin{bmatrix} \zeta_{j,t}^k & \zeta_{j,t}^g & \zeta_{j,t}^s & \zeta_{j,t}^a \end{bmatrix}$ captures the market prices of risk for each Brownian shock in $\boldsymbol{dZ_t}$. Since both agents trade in the risk-free market, they receive the same return r_t. Following Gopalakrishna [2020], I assume that the experts are subjected to Poisson shocks that will force them to exit the economy and become households. That is, at each time instant dt, a fraction $\tau_t dt$ of the experts transition into households.

[19] The investment rate can also be different between the agents, but it turns out that in equilibrium, the optimal investment is tightly linked to the capital price. Since the price is unique, the investment rate $\iota_{j,t}$ is the same for all agents.

[20] The return for agent j is $dR_{j,t} = \frac{(a_{j,t} - \iota_{j,t})k_{j,t}}{q_t k_{j,t}} dt + \frac{d(q_t k_{j,t})}{q_t k_{j,t}}$

Equilibrium: The optimization problem for each agent type j is given by

$$U_{j,t} = \sup_{c_{j,t},k_{j,t}} E_t\left[\int_t^{T_j} f(c_{j,s},U_{j,s})ds + 1_{j\in\mathbb{E}}U_{h,\tau'}\right] \tag{46}$$

$$\text{s.t.}\frac{dw_{j,t}}{w_{j,t}} = \left(r_t - \frac{c_{j,t}}{w_{j,t}} + \frac{q_t k_{j,t}}{w_{j,t}}((\mu^R_{j,t} - r_t) - (1-\chi_{j,t})\bar{\epsilon}_{j',t})\right)dt + \sigma_{w_j,t}(\boldsymbol{\sigma^R_t})^T\boldsymbol{dZ_t} \qquad j\in\{e,h\}$$

where $\bar{\epsilon}_{j,t} = \boldsymbol{\zeta}^T_{j,t}\boldsymbol{\sigma^R_t}$, $T_j = \tau'$ for experts and $T_j = \infty$ for households, and τ' is the time of transition. The agent j earns $\mu_{j,t} - r_t$ from borrowing in the risk free market and investing in the risky capital, but have to pay the outside equity holders a compensation for their risk. Households do not issue outside equity (i.e., $\chi_{h,t} = 1$) and I denote $\chi_{e,t}$ as χ_t for simplicity of notation. The volatility terms in wealth is given by

$$\sigma_{w_e,t} = \frac{q_t k_{e,t}}{w_{e,t}}\chi_t \tag{47}$$

$$\sigma_{w_h,t} = \frac{q_t k_{h,t}}{w_{h,t}} + (1-\chi_t)\frac{q_t w_{e,t}}{w_{h,t}} \tag{48}$$

The experts retain the fraction χ_t of risk in their balance sheet and offload the remaining to the households. The agents solve for the optimal consumption $c_{j,t}$ and portfolio holdings $k_{j,t}$ by maximizing the objective function (46). The optimal investment rate $\iota_{j,t}$ is found by maximizing the expected return on risky capital. The competitive equilibrium is defined as

Definition 5.1. A competitive equilibrium is a set of aggregate stochastic processes adapted to the filtration generated by the Brownian motion $\boldsymbol{Z_t}$. Given an initial distribution of wealth between the experts and the households, the processes are prices (q_t, r_t), policy functions $(c_{j,t}, \iota_{j,t}, k_{j,t}; j\in\{e,h\})$, and net worth $(w_{j,t}; j\in\{e,h\})$, such that

- Capital market clears: $\int_{\mathbb{H}}(1-\psi_t)K_t dj + \int_{\mathbb{E}}\psi_t K_t dj = \int_{\mathbb{H}\cup\mathbb{E}} k_{j,t}dj \quad \forall t$
- Goods market clear: $\int_{\mathbb{H}\cup\mathbb{E}} c_{j,t}dj = \int_{\mathbb{H}\cup\mathbb{E}}(a_{j,t} - \iota_{j,t})k_{j,t}dj \quad \forall t$
- $\int_{\mathbb{H}\cup\mathbb{E}} w_{j,t}dj = \int_{\mathbb{H}\cup\mathbb{E}} q_t k_{j,t}dj \quad \forall t$

I seek a Markov equilibrium where each agents within the same type compute the

optimal policies. Let the wealth share experts be defined by

$$z_t = \frac{W_{e,t}}{q_t K_t} \qquad \in (0,1)$$

where $W_{e,t} = \int_{\mathbb{E}} w_{j,t} dj$ and $K_t = \int_{\mathbb{E}} k_{j,t} dj + \int_{\mathbb{H}} k_{j,t} dj$. The wealth share z_t is the endogenous state variable in the model which moves in response to the other equilibrium objects. The set of exogenous state variables is given by $\{g_t, s_t, a_{e,t}\}$. These four state variables characterize the whole system. The stochastic processes for the exogenous state variables are given in (38), (39), and (37) respectively. The proposition below provides the process for the endogenous state variable.

Proposition 1. *The law of motion of the wealth share of experts is given by*

$$\frac{dz_t}{z_t} = \mu_t^z dt + (\boldsymbol{\sigma_t^z})^T \boldsymbol{dZ_t} \tag{49}$$

where

$$\mu_t^z = \frac{a_{e,t} - \iota_{e,t}}{q_t} - \frac{C_{e,t}}{W_{e,t}} + \frac{\chi_t \psi_t}{z_t}(\tilde{\boldsymbol{\zeta}}_{e,t}^T \boldsymbol{\sigma_t^R}) + (1-\chi_t)(\boldsymbol{\zeta}_{e,t}^T - \boldsymbol{\zeta}_{h,t}^T)\boldsymbol{\sigma_t^R} - \tau_t$$

$$(\boldsymbol{\sigma_t^z}) := \begin{bmatrix} \sigma_t^{z,k} & \sigma_t^{z,g} & \sigma_t^{z,s} & \sigma_t^{z,a} \end{bmatrix}^T = (\frac{\chi_t \psi_t}{z_t} - 1)\boldsymbol{\sigma_t^R}$$

$$\tilde{\boldsymbol{\zeta}}_{e,t}^T := (\boldsymbol{\sigma_t^R})^T (\boldsymbol{\zeta}_{e,t} - \boldsymbol{\sigma_t^R})$$

Proof: See Appendix A.3.2.

Note that the exit rate τ_t enters the drift of the wealth share.

Asset pricing conditions: The agents maximize the return on capital to obtain the optimal investment rate. That is, they solve

$$\max_{\iota_{j,t}} \quad \Phi(\iota_{j,t}) - \frac{\iota_{j,t}}{q_t}$$

I assume a simple investment cost function given by $\Phi(\iota) = \frac{\log(\kappa\iota + 1)}{\kappa}$. Then, $\iota_{j,t}^*$ is given by

$$\iota_{j,t}^* = \frac{q_t - 1}{\kappa} \tag{50}$$

where κ is an investment cost parameter. Since the optimal investment rate depends only on the capital price, it is not agent-specific. This is a reflection of q-theory result which ties the investment rate and capital price tightly. The asset pricing condition for the experts is given by[21]

$$\frac{a_{e,t} - \iota_t}{q_t} + \Phi(\iota_t) - \delta + g_t + \mu_t^q + \sigma\sqrt{s_t}\sigma_t^{q,k} - r_t = \chi_t \bar{\boldsymbol{\epsilon}}_{e,t} + (1 - \chi_t)\bar{\boldsymbol{\epsilon}}_{h,t} \tag{51}$$

where $\bar{\epsilon}_{j,t} = \boldsymbol{\zeta}_{j,t}^T \boldsymbol{\sigma}_t^{\boldsymbol{R}}$ and χ_t is the share of inside equity chosen by the experts. For the households, the condition is given by

$$\frac{a_h - \iota_t}{q_t} + \Phi(\iota_t) - \delta + g_t + \mu_t^q + \sigma\sqrt{s_t}\sigma_t^{q,k} - r_t \leq \bar{\boldsymbol{\epsilon}}_{h,t} \tag{52}$$

with equality when $\psi_t < 1$. We can combine the asset pricing condition for experts and households to get

$$\frac{a_{e,t} - a_h}{q_t} \geq \chi_t(\bar{\epsilon}_{e,t} - \bar{\epsilon}_{h,t}) \tag{53}$$

$$\max\{\underline{\chi} - \chi_t, \bar{\epsilon}_{e,t} - \bar{\epsilon}_{h,t}\} = 0 \tag{54}$$

where (53) holds with equality if risk premia of experts is larger than that of the households. In regions of the state space where wealth share is sufficiently large, χ_t is chosen to satisfy $\bar{\epsilon}_{e,t} = \bar{\epsilon}_{h,t}$.

HJB equations: The HJB problem for agent type j can be written as

$$\sup_{c_{j,t}, k_{j,t}} f(c_{j,t}, U_{j,t}) + E[dU_{j,t}] = 0 \tag{55}$$

The conjecture for value function is of the form

$$U_{j,t} = \frac{(J_{j,t}(z_t, g_t, s_t, a_{e,t})K_t)^{1-\gamma}}{1-\gamma}$$

[21] This can be derived using a martingale argument as shown in Appendix A.3.1.

where $J_{j,t}(\cdot)$ captures the stochastic investment opportunity set. The process for $J_{j,t}$ is given by

$$\frac{\partial J_{j,t}}{J_{j,t}} = \mu^J_{j,t} dt + \boldsymbol{\sigma}^{\boldsymbol{J}}_{j,t} \boldsymbol{dZ_t}$$

where the quantities $\mu^J_{j,t}$ and $\boldsymbol{\sigma}^{\boldsymbol{J}}_{j,t} = \begin{bmatrix} \sigma^k_{j,t} & \sigma^g_{j,t} & \sigma^s_{j,t} & \sigma^a_{j,t} \end{bmatrix}$ are to be solved in equilibrium.

Proposition 2. *The optimal policies are given by*

$$\hat{c}_{j,t} = \rho \tag{56}$$

$$\zeta^k_{j,t} = -\sigma^{J,k}_t + \sigma^{z,k}_{j,t} + \sigma^{q,k}_t + \gamma\sigma\sqrt{s_t} \tag{57}$$

$$\zeta^i_{j,t} = -\sigma^{J,i}_{j,t} + \sigma^{z,i}_{j,t} + \sigma^{q,i}_t \qquad i \in \{g, s, a\} \tag{58}$$

Proof: See Appendix.

Since the IES is set to 1, the consumption-wealth ratio ($\hat{c}_{j,t}$) is equal to the discount rate ρ. The market prices of risk are given up to the other equilibrium objects $(\mu^J_{j,t}, \boldsymbol{\sigma}^{\boldsymbol{J}}_{j,t}, \boldsymbol{\sigma}^{\boldsymbol{R}}_{\boldsymbol{t}}, q_t, \psi_t)$ which are solved in the state space $\boldsymbol{x}^T_t = \begin{bmatrix} z_t & g_t & s_t & a_{e,t} \end{bmatrix}$.

Definition 5.2. A Markov equilibrium in $(\mathbf{z_t} \in (\mathbf{0}, \mathbf{1}), \mathbf{g_t} \in (\underline{\mathbf{g}}, \mathbf{g}), \mathbf{s_t} \in (\underline{\mathbf{s}}, \mathbf{s}), \mathbf{a_{e,t}} \in (\underline{\mathbf{a}}_\mathbf{e}, \mathbf{a_e}))$ is a set of adapted processes $q(\boldsymbol{x_t}), r(\boldsymbol{x_t}), J_e(\boldsymbol{x_t}), J_h(\boldsymbol{x_t})$, policy functions $\hat{c}_e(\boldsymbol{x_t}), \hat{c}_h(\boldsymbol{x_t}), \psi(\boldsymbol{x_t})$, and state variables $\boldsymbol{x_t}$ such that

- $J_{j,t}$ solves the HJB equations and corresponding policy functions $\hat{c}_{j,t}, \psi_t$
- Markets clear

$$(\hat{c}_{e,t} z_t + \hat{c}_{h,t}(1 - z_t)) q_t = \psi_t (a_{e,t} - \iota_t) + (1 - \psi_t)(a_h - \iota_t)$$

$$w_{e,t} z_t + w_{h,t}(1 - z_t) = 1$$

- The state variables $\boldsymbol{x_t}$ satisfy (38), (39), (37), and (86).

5.2 Numerical method

The model is solved numerically that involves two blocks. The first block is the static inner step that takes $J_{j,t}$ as given and solves for the equilibrium quantities $(q_t, \chi_t, \psi_t, \boldsymbol{\sigma}^{\boldsymbol{R}}_{\boldsymbol{t}})$. The other equilibrium objects are computed using these quantities. The second block

is the outer time step that takes the equilibrium quantities as given and updates the values of $J_{j,t}$ by solving a de-coupled system of PDEs- one for the expert and one for the household. The inner block is solved using a Newton-Rhaphson method that is computationally fast, and the outer block is solved using ALIENs to overcome the curse of dimensionality problem.

Proposition 3. *The equilibrium objects* $(\psi_t, q_t, \sigma_t^{q,k}, \sigma_{q,g}, \sigma_{q,s}, \sigma_{q,a})$ *are found by solving the differential-algebraic system of equations given by*

$$0 = \frac{a_{e,t} - a_h}{q_t} - \chi_t \left((1-\gamma) \left(\frac{1}{J_{h,t}} \frac{\partial J_{h,t}}{\partial z_t} - \frac{1}{J_{e,t}} \frac{\partial J_{e,t}}{\partial z_t} \right) + \frac{1}{z_t(1-z_t)} \right) (\chi_t \psi_t - z_t)[||\boldsymbol{\sigma^R}||^2]$$
$$- \sum_{i \in \{g,s,a\}} \chi_t (1-\gamma) \left(\frac{1}{J_{h,t}} \frac{\partial J_{h,t}}{\partial i_t} - \frac{1}{J_{e,t}} \frac{\partial J_{e,t}}{\partial i_t} \sigma^i \sigma^{q,i} \right) \quad if \quad \psi_t < 1 \tag{59}$$

$$0 = (\rho_e z_t + (1-z_t)\rho_h) q_t - \psi_t (a_{e,t} - \iota_t) + (1-\psi_t)(a_h - \iota_t) \tag{60}$$

$$0 = \sigma_t^{q,k} + \sigma \sqrt{s_t} - \frac{\sigma \sqrt{s_t}}{1 - \frac{1}{q_t} \frac{\partial q_t}{\partial z_t} (\chi_t \psi_t - z_t)} \tag{61}$$

$$0 = \sigma_t^{q,i} - \frac{\frac{1}{q_t} \frac{\partial q_t}{\partial i_t}}{1 - \frac{1}{q_t} \frac{\partial q_t}{\partial z_t} (\chi_t \psi_t - z_t)} \qquad i \in \{g,s,a\} \tag{62}$$

Proof: See Appendix.

The first equation is obtained from the differences in risk premium of the experts and the households from equations (51) and (52). The second equation comes from the goods market clearing condition and the remaining equations are obtained from writing return volatility objects in terms of other equilibrium quantities. I employ a Newton-Rhaphson method that is computationally fast even in high dimensions. The Newton method is highly sensitive to the initial values provided. To avoid errors, I provide as inputs the equilibrium values from prior points in the grid. I relegate further details to the Appendix. Once the six quantities are found, the other other equilibrium objects $(\boldsymbol{\zeta}_{e,t}, \boldsymbol{\zeta}_{h,t}, \mu_t^z, \boldsymbol{\sigma_t^z}, \mu_{e,t}^J, \mu_{h,t}^J, \boldsymbol{\sigma}_{e,t}^{\boldsymbol{J}}, \boldsymbol{\sigma}_{h,t}^{\boldsymbol{J}})$ are easily computed since they are a function of the state variables and the other variables found in the static step.

Proposition 4. *The stochastic opportunity set J is characterized by the PDE*[22]

$$0 = \frac{\partial J}{\partial t} + A\left(\boldsymbol{x}, J, \frac{\partial J}{\partial \boldsymbol{x}}\right) + \frac{1}{2} tr\left[B\left(\boldsymbol{x}, J, \frac{\partial J}{\partial \boldsymbol{x}}\right) \frac{\partial^2 J}{\partial \boldsymbol{x}^2} B\left(\boldsymbol{x}, J, \frac{\partial J}{\partial \boldsymbol{x}}\right)^T\right] \tag{63}$$

with the boundary conditions

$$J(\boldsymbol{x}, t) = \tilde{J} \tag{64}$$

$$\partial J(\boldsymbol{x}^0, t) = \partial J(\boldsymbol{x}^1, t) = 0 \tag{65}$$

where[23]

$$\begin{aligned}
A\left(\boldsymbol{x}, J, \frac{\partial J}{\partial \boldsymbol{x}}\right) &= J\Big(\rho\big(\log\rho - \log J + \log(qz)\big) + \Phi(\iota_t) + g_t - \delta\Big) \\
&\quad - J\left((\gamma - 1)\sigma\sigma^{J,k} + \frac{\gamma}{2}||\sigma^J||^2 - 1_{j\in\mathbb{E}}\frac{\tau_t}{1-\gamma}\left(\left(\frac{J_{j,t}}{J_{j',t}}\right)^{1-\gamma} - 1\right)\right) \\
&\quad + \begin{bmatrix} \frac{\partial J}{\partial z} & \frac{\partial J}{\partial g} & \frac{\partial J}{\partial s} & \frac{\partial J}{\partial a_e} \end{bmatrix} \begin{bmatrix} z\mu^z & \mu^g & \mu^s & \mu^a \end{bmatrix}^T \\
B\left(\boldsymbol{x}, J, \frac{\partial J}{\partial \boldsymbol{x}}\right) &= \begin{bmatrix} \sigma^z & \sigma^g & \sigma^s & \sigma^a \end{bmatrix} \\
\frac{\partial^2 J}{\partial \boldsymbol{x}^2} &= \begin{bmatrix} \frac{\partial^2 J}{\partial z^2} & \frac{\partial^2 J}{\partial g^2} & \frac{\partial^2 J}{\partial s^2} & \frac{\partial^2 J}{\partial a_e^2} \end{bmatrix} \\
\boldsymbol{x}^0 &= \{(0, g, s, a), (z, \underline{g}, s, a), (z, g, \underline{s}, a), (z, g, s, \underline{a}_e)\} \\
\boldsymbol{x}^1 &= \{(1, g, s, a), (z, \bar{g}, s, a), (z, g, \bar{s}, a), (z, g, s, \bar{a}_e)\}
\end{aligned}$$

Proof: See Appendix.

Once the static step is completed, it remains to solve the partial differential equations given in (63). The coefficients of the PDE are computed using the solution form static step. Hence, taking the coefficients as given, the task is to find the function J that solves the above PDE.

5.3 Solution

Figure (5) presents the equilibrium quantities for different values of volatility. A higher volatility s_t leads to lower capital price throughout the state space since it reflects an

[22] I write $J_{j,t}$ simply as J to avoid clustering of notations.

[23] The index j' refers to the other type of agent.

increased macroeconomic uncertainty. Interestingly, a lower volatility decreases the endogenous return volatility in the normal region but increases the return volatility in the crisis region. In Brunnermeier and Sannikov [2014], the volatility is constant but a static comparison leads to similar result, which they call as the 'volatility paradox'. The paradox refers to the fact that a decrease in exogenous fundamental volatility leads to an increase in endogenous price volatility during crises. This paradox carries over to a more general model of stochastic volatility as well. The volatility has a decreasing effect on the drift of the wealth share. A higher volatility reduces the risk premium and capital price and therefore leads to a lower drift. This has important implications on how the system transitions in and out of crises. A larger volatility means that the system spends longer in crises since the experts rebuild their wealth slowly due to reduced risk premium. In the normal region where experts are wealthy, a higher volatility increases the risk premium and hence the drift of wealth share is larger. Figure (6) presents the equilibrium quantities for different values of expert productivity $a_{e,t}$. The time variation induced by productivity on the risk prices are higher than that induced by volatility, since the former directly affects the capital price through the goods market clearing condition. When experts are more productive, the capital price is larger since the aggregate dividend is increasing in the proportion of capital held by the more productive experts. When productivity drops towards the lowest level of 10%, the risk premium goes down decreasing the drift of the wealth share. Hence, less productive experts force the system to spend a longer time in crises. As productivity revers to its upper level of 20%, the drift of the wealth share increases and pushes the system out of crises. Gopalakrishna [2020] shows a similar phenomenon where the twin forces of stochastic productivity and regime-dependent transition rate of experts help quantitatively explain the crisis dynamics.

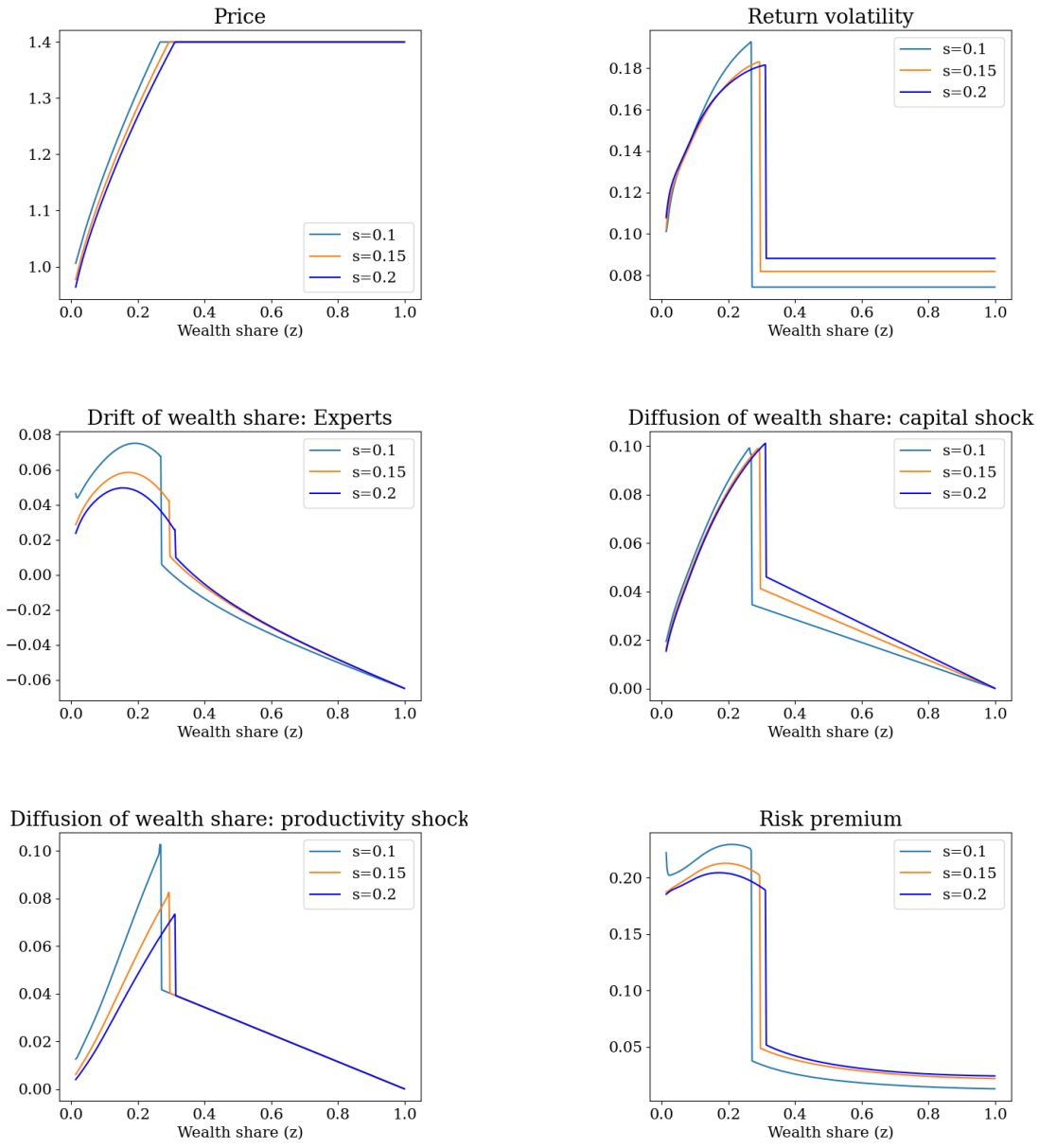

Figure 5. Equilibrium quantities for different volatility values (s_t). Growth rate (g_t) and productivity ($a_{e,t}$) are fixed at respective average values.

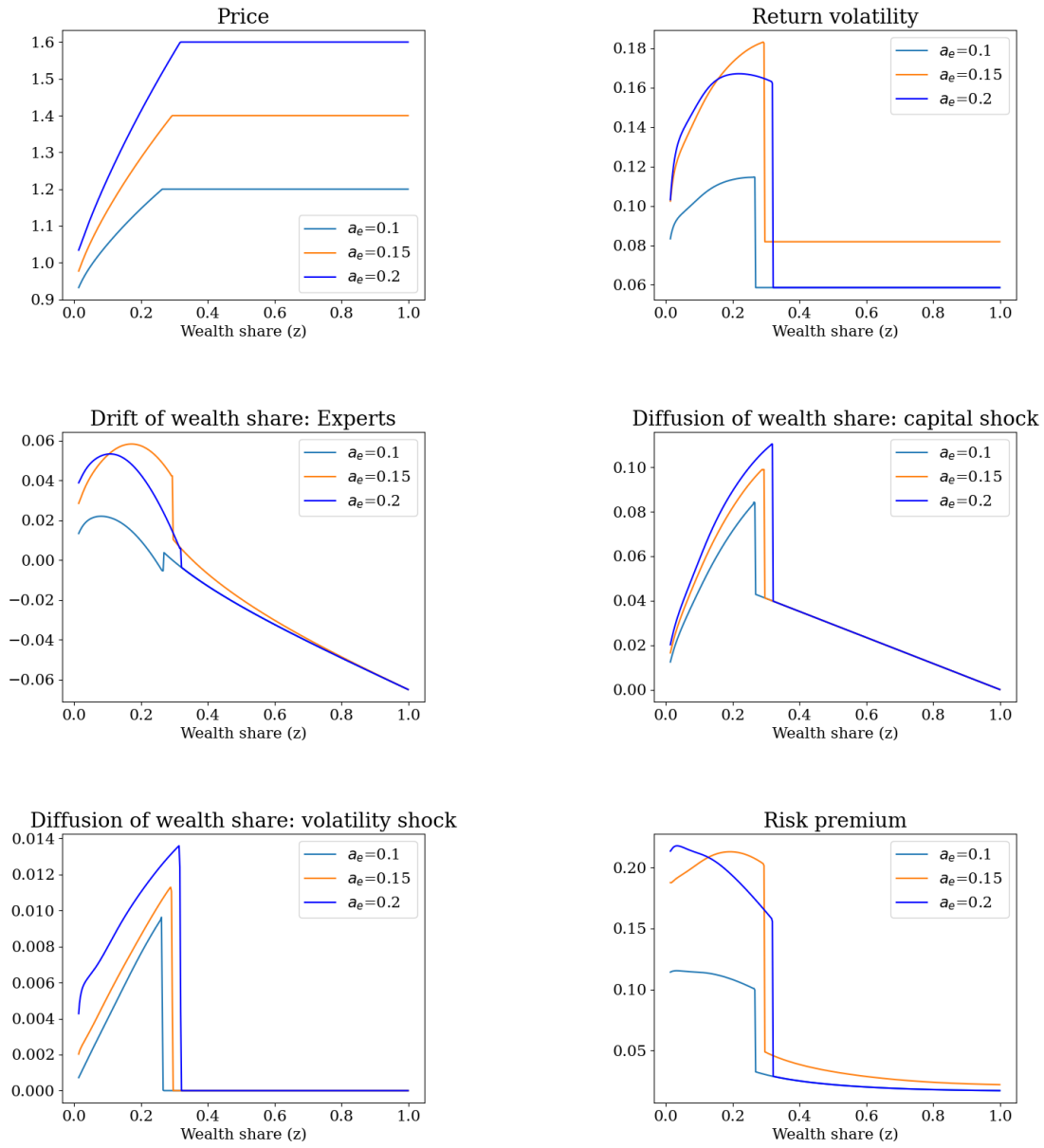

Figure 6. Equilibrium quantities for different productivity values ($a_{e,t}$). Growth rate (g_t) and volatility (s_t) are fixed at respective average values.

6 Conclusion

I have developed a new computational technique called Actively Learned and Informed Equilibrium Nets (ALIENs) to solve continuous time economic models featuring heterogeneous agents, occasionally binding constraints, endogenous jumps, and highly non-linear policy functions. The technique relies on solving a system of parabolic differential equations using a collection of deep neural networks by converting them into a sequence of supervised learning problems. The HJB equation from the agents' optimization problem takes the form of a non-linear elliptical PDE, which is converted into a quasi-linear parabolic type by adding an artificial time derivative and treating the PDE coefficients as known constants. The value function that solves the PDE is approximated using a neural network and is forced to obey the laws that are governed by the economic system. The architecture is split into i) a PDE network that is responsible for the fitted value function to satisfy the partial differential equation, ii) a boundary network that strives to fit the boundary condition of the PDE, and iii) an active network that ensures a better fit in the state space with most economic information. I utilize data parallelism that leverages the computing hardware to significantly speed up the computational time.

I have applied the method to solve a macro-finance benchmark model with an endogenous state variable, non-linear policy functions, and showed that the neural network solution matches the finite difference solution. The second application is to a similar model with capital misallocation, endogenous jumps, and endogenous state variables but in a higher dimension. The value function in this problem is challenging to solve using a traditional finite difference scheme, since endogenous PDE coefficients lead to difficulty in maintaining the monotonicity of the scheme. In addition, the high dimensionality of PDE creates a massive computational bottleneck. ALIENs successfully sidesteps these limitations and ensures convergence by actively tracking the subdomain with most economic information to create informed sample points for training the neural network. Lots of problems in continuous time are formulated in a small dimensional state space due to computational bottlenecks. The ease with which ALIENs can be implemented opens up opportunities in fields as diverse as macroeconomics, asset pricing, and dynamic corporate finance. Moreover, the fact that ALIENs are powered with data parallelism through minimal effort opens up new avenues to

perform extensive quantitative analysis that requires experimentation and repeatedly solving models several times.

References

Yves Achdou, Francisco J. Buera, Jean-Michel Lasry, Pierre-Louis Lions, and Benjamin Moll. PDE Models in Macroeconomics. *Proceedings of the Royal Society*, 2014a.

Yves Achdou, Jiequn Han, Jean-Michel Lasry, Pierre-Louis Lions, and Benjamin Moll. Heterogeneous agent models in continuous time. *Preprint*, 2014b.

Yves Achdou, Jiequn Han, Jean-Michel Lasry, Pierre-Louis Lions, and Benjamin Moll. Income and wealth distribution in macroeconomics: A continuous-time approach. *NBER Working Paper Series*, 2017. doi: 10.3386/w23732.

Tobias Adrian and Nina Boyarchenko. Intermediary Leverage Cycles and Financial Stability. *SSRN Electronic Journal*, (August 2012), 2012. doi: 10.2139/ssrn.2133385.

Tobias Adrian and Nina Boyarchenko. Liquidity policies and systemic risk. *Journal of Financial Intermediation*, 2018. ISSN 10960473. doi: 10.1016/j.jfi.2017.08.005.

Andrew Gibiansky. Bringing HPC techniques to deep learning. *http://research.baidu.com/bringing-hpc-techniques-deep-learning. [Online; accessed 10-June-2019].*, 2017.

Marlon Azinovic, Luca Gaegauf, and Simon Scheidegger. Deep Equilibrium Nets. *SSRN Electronic Journal*, 2019. ISSN 1556-5068. doi: 10.2139/ssrn.3393482.

Ravi Bansal and Amir Yaron. Risks for the long run: A potential resolution of asset pricing puzzles, 2004. ISSN 00221082.

Suleyman Basak and Domenico Cuoco. An equilibrium model with restricted stock market participation. *Review of Financial Studies*, 1998. ISSN 08939454. doi: 10.1093/rfs/11.2.309.

Suleyman Basak and Alexander Shapiro. Value-at-risk-based risk management: Optimal policies and asset prices. *Review of Financial Studies*, 2001. ISSN 08939454. doi: 10.1093/rfs/14.2.371.

Ben S Bernanke, Ben S Bernanke, Mark Gertler, Mark Gertler, Simon Gilchrist, and Simon Gilchrist. The Financial Accelerator in a Business Cycle Framework. *NBER Working Paper*, 1998. ISSN 15740048.

Patrick Bolton, Hui Chen, and Neng Wang. A Unified Theory of Tobin'sq, Corporate Investment, Financing, and Risk Management. *Journal of Finance*, 2011. ISSN 00221082. doi: 10.1111/j.1540-6261.2011.01681.x.

Patrick Bolton, Hui Chen, and Neng Wang. Market timing, investment, and risk management. *Journal of Financial Economics*, 2013. ISSN 0304405X. doi: 10.1016/j.jfineco.2013.02.006.

J. Frédéric Bonnans, Élisabeth Ottenwaelter, and Housnaa Zidani. A fast algorithm for the two dimensional HJB equation of stochastic control. *Mathematical Modelling and Numerical Analysis*, 2004. ISSN 0764583X. doi: 10.1051/m2an:2004034.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. ISSN 23318422.

Johannes Brumm and Simon Scheidegger. Using Adaptive Sparse Grids to Solve High-Dimensional Dynamic Models. *Econometrica*, 2017. ISSN 0012-9682. doi: 10.3982/ecta12216.

M. K. Brunnermeier and Y. Sannikov. Macro, Money, and Finance: A Continuous-Time Approach. In *Handbook of Macroeconomics*. 2016. ISBN 9780444594877. doi: 10.1016/bs.hesmac.2016.06.002.

Markus K Brunnermeier and Yuliy Sannikov. A macroeconomic model with a financial sector. *American Economic Review*, 104(2):379–421, 2014. ISSN 00028282. doi: 10.1257/aer.104.2.379.

Hans Joachim Bungartz, Alexander Heinecke, Dirk Pflüger, and Stefanie Schraufstetter. Option pricing with a direct adaptive sparse grid approach. In *Journal of Computational and Applied Mathematics*, 2012. doi: 10.1016/j.cam.2011.09.024.

Ricardo J. Caballero and Alp Simsek. A Risk-Centric Model of Demand Recessions and Macroprudential Policy. *SSRN Electronic Journal*, 2017. ISSN 1556-5068. doi: 10.2139/ssrn.3004727.

Luyang Chen, Markus Pelger, and Jason Zhu. Deep Learning in Asset Pricing. *SSRN Electronic Journal*, 2019. ISSN 1556-5068. doi: 10.2139/ssrn.3350138.

Adrien D'Avernas and Quentin Vandeweyer. A Solution Method for Continuous-Time Models. pages 1–33, 2019.

Adrien D'Avernas, Quentin Vandeweyer, and Matthieu Darracq Paries. Unconventional Monetary Policy and Funding Liquidity Risk. *SSRN Electronic Journal*, 2019. ISSN 1556-5068. doi: 10.2139/ssrn.3500556.

Sebastian Di Tella. Uncertainty shocks and balance sheet recessions. *Journal of Political Economy*, 125(6):2038–2081, 2017. ISSN 1537534X. doi: 10.1086/694290.

Sebastian Di Tella. Optimal regulation of financial intermediaries. *American Economic Review*, 2019. ISSN 19447981. doi: 10.1257/aer.20161488.

Sebastian Di Tella. Risk premia and the real effects of money. *American Economic Review*, 2020. ISSN 19447981. doi: 10.1257/aer.20180203.

Itamar Drechsler, Alexi Savov, and Philipp Schnabl. The deposits channel of monetary policy. *Quarterly Journal of Economics*, 2017. ISSN 15314650. doi: 10.1093/qje/qjx019.

Itamar Drechsler, Alexi Savov, and Philipp Schnabl. A Model of Monetary Policy and Risk Premia. *Journal of Finance*, 2018. ISSN 15406261. doi: 10.1111/jofi.12539.

Victor Duarte. Machine Learning for Continuous-Time Economics. 2017.

Jesús Fernández-Villaverde, Samuel Hurtado, and Galo Nuno. Financial Frictions and the Wealth Distribution. *SSRN Electronic Journal*, 2020. ISSN 1556-5068. doi: 10.2139/ssrn.3615695.

Nicolae Gârleanu and Stavros Panageas. Young, old, conservative, and bold: The implications of heterogeneity and finite lives for asset pricing. *Journal of Political Economy*, 2015. ISSN 1537534X. doi: 10.1086/680996.

Nicolae Gârleanu and Lasse Heje Pedersen. Margin-based asset pricing and deviations from the law of one price. *Review of Financial Studies*, 2011. ISSN 08939454. doi: 10.1093/rfs/hhr027.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Journal of Machine Learning Research*, 2010a.

Xavier Glorot and Yoshua Bengio. Xavier Initialization. *Journal of Machine Learning Research*, 2010b. ISSN 15324435.

Matthieu Gomez. Asset Prices and Wealth Inequality. *Working Paper*, 2019.

Goutham Gopalakrishna. A Macro-Finance model with Realistic Crisis Dynamics. *SSRN Electronic Journal*, 2020. ISSN 1556-5068. doi: 10.2139/ssrn.3732232.

Denis Gromb and Dimitri Vayanos. Equilibrium and welfare in markets with financially constrained arbitrageurs. *Journal of Financial Economics*, 2002. ISSN 0304405X. doi: 10.1016/s0304-405x(02)00228-3.

Shihao Gu, Bryan Kelly, and Dacheng Xiu. Autoencoder asset pricing models. *Journal of Econometrics*, 2020. ISSN 18726895. doi: 10.1016/j.jeconom.2020.07.009.

Valentin Haddad. Concentrated Ownership and Equilibrium Asset Prices. *SSRN Electronic Journal*, 2012. doi: 10.2139/ssrn.2140861.

Jiequn Han, Arnulf Jentzen, and E. Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences of the United States of America*, 2018. ISSN 10916490. doi: 10.1073/pnas.1718942115.

Lars Peter Hansen, Paymon Khorrami, and Fabrice Tourre. Comparative Valuation Dynamics in Models with Financing Restrictions. (February), 2018. URL http://www.zccfe.uzh.ch/dam/jcr:c4a09093-fc00-4953-9f31-701dc0cd3fab/macro-model-print.pdf.

Zhiguo He and Arvind Krishnamurthy. Intermediary asset pricing. *American Economic Review*, 103(2):732–770, 2013. ISSN 00028282. doi: 10.1257/aer.103.2.732.

Zhiguo He and Arvind Krishnamurthy. A macroeconomic framework for quantifying systemic risk. *American Economic Journal: Macroeconomics*, 11(4):1–37, 2019. ISSN 19457715. doi: 10.1257/mac.20180011.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 1991. ISSN 08936080. doi: 10.1016/0893-6080(91)90009-T.

Kenneth L Judd. Projection methods for solving aggregate growth models. *Journal of Economic Theory*, 1992. ISSN 10957235. doi: 10.1016/0022-0531(92)90061-L.

Susan E. Kelly. Gibbs phenomenon for wavelets. *Applied and Computational Harmonic Analysis*, 1996. ISSN 10635203. doi: 10.1006/acha.1996.0006.

Nobuhiro Kiyotaki and John Moore. Credit cycles. *Journal of Political Economy*, 1997. ISSN 00223808. doi: 10.1086/262072.

Arvind Krishnamurthy and Wenhao Li. Dissecting Mechanisms of Financial Crises: Intermediation and Sentiment. *SSRN Electronic Journal*, 2020. ISSN 1556-5068. doi: 10.2139/ssrn.3554788.

Per Krusell and Anthony A. Smith. Income and wealth heterogeneity in the macroeconomy. *Journal of Political Economy*, 1998. ISSN 00223808. doi: 10.1086/250034.

Pablo Kurlat. How I Learned to Stop Worrying and Love Fire Sales. *National Bureau of Economic Research*, 2018. ISSN 0898-2937. doi: 10.3386/w24752.

Deep Learning. Deep Learning - Goodfellow. *Nature*, 2016.

Wenhao Li. Public liquidity, bank runs, and financial crises. *Working Paper*, 2020.

Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations, 2019. ISSN 23318422.

G. D. Mallinson and G. de Vahl Davis. The method of the false transient for the solution of coupled elliptic equations. *Journal of Computational Physics*, 1973. ISSN 10902716. doi: 10.1016/0021-9991(73)90097-1.

Tobias J. Moskowitz, Yao Hua Ooi, and Lasse Heje Pedersen. Time series momentum. *Journal of Financial Economics*, 2012. ISSN 0304405X. doi: 10.1016/j.jfineco.2011.11.003.

M Raissi, P Perdikaris, and G E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 2019. ISSN 10902716. doi: 10.1016/j.jcp.2018.10.045.

Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *Journal of Machine Learning Research*, 2018. ISSN 15337928.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (Part II): Data-driven discovery of nonlinear partial differential equations, 2017. ISSN 23318422.

Philipp Johannes Renner and Simon Scheidegger. Machine Learning for Dynamic Incentive Problems. *SSRN Electronic Journal*, 2018. ISSN 1556-5068. doi: 10.2139/ssrn.3282487.

Simon Scheidegger and Ilias Bilionis. Machine learning for high-dimensional dynamic stochastic economies. *Journal of Computational Science*, 2019. ISSN 18777503. doi: 10.1016/j.jocs.2019.03.004.

Alexander Sergeev and Mike Del Balso. Horovod: Fast and easy distributed deep learning in tensorflow, 2018a. ISSN 23318422.

Alexander Sergeev and Mike Del Balso. Horovod: Fast and easy distributed deep learning in tensorflow, 2018b. ISSN 23318422.

Dejanir Silva. The Risk Channel of Unconventional Monetary Policy. *University of Illinois at Urbana-Champaign Working Paper*, 2020.

Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 2018a. ISSN 10902716. doi: 10.1016/j.jcp.2018.08.029.

Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 2018b. ISSN 10902716. doi: 10.1016/j.jcp.2018.08.029.

Sebastian Di Tella and Pablo Kurlat. Why are Banks Exposed to Monetary Policy? *National Bureau of Economic Research*, 2017a. ISSN 0898-2937. doi: 10.3386/w24076.

Sebastian Di Tella and Pablo Kurlat. Why are Banks Exposed to Monetary Policy? *National Bureau of Economic Research*, 2017b. ISSN 0898-2937. doi: 10.3386/w24076.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *34th International Conference on Machine Learning, ICML 2017*, 2017. ISBN 9781510855144.

Jessica A. Wachter. Can Time-Varying Risk of Rare Disasters Explain Aggregate Stock Market Volatility? *Journal of Finance*, 2013. ISSN 00221082. doi: 10.1111/jofi.12018.

A Appendix

A.1 Proof of (3.1)

I consider a special case of quasi-linear parabolic PDE of the form[24]

$$\mathcal{G}[J](t,\boldsymbol{x}) := \frac{\partial J}{\partial t} + \sum_{j}^{d} a(t,\boldsymbol{x})\frac{\partial J}{\partial x_j} + \sum_{j}^{d} b(t,\boldsymbol{x})\frac{\partial^2 J}{\partial x_i \partial x_j} + \hat{A}(t,\boldsymbol{x},J(t,\boldsymbol{x}),\nabla J(t,\boldsymbol{x})) = 0; \qquad \forall (t,\boldsymbol{x}) \in \Omega_T \tag{66}$$

with the boundary conditions

$$J(t,\boldsymbol{x}) = J_0(t,\boldsymbol{x}); \quad \forall (t,\boldsymbol{x}) \in \Omega \tag{67}$$

$$J(t,\boldsymbol{x}) = J_0(t,\boldsymbol{x}); \quad \forall (t,\boldsymbol{x}) \in \Omega_c \tag{68}$$

$$\mathcal{G}[J](t,\boldsymbol{x}) = 0; \qquad \forall (t,\boldsymbol{x}) \in \Omega_c \tag{69}$$

$$\frac{\partial J(t,\boldsymbol{x})}{\partial \boldsymbol{x}} = 0; \quad \forall (t,\boldsymbol{x}) \in \partial\Omega_T \tag{70}$$

Note that the functions $a(t,\boldsymbol{x})$ and $b(t,\boldsymbol{x})$ are bounded and do not depend on the function J. This is a valid assumption since the PDE in (12) is solved using value function iteration where the coefficients are determined in the inner static loop and hence do not depend on the function J that needs to be solved.[25] I also assume that the PDE (66) has a unique solution $J(t,\boldsymbol{x}) \in C^{1,2}(\Omega_T)$ with its derivatives uniformly bounded. I prove the Theorem (3.1) for this special class of quasi-linear parabolic PDE, that is common in the macro-finance literature (see Achdou et al. [2017]), using a single hidden layer neural network with n number of neurons. Consider the class

$$\check{C}^n(\sigma) = \left\{ h(t,\boldsymbol{x}) : \mathbb{R}^{1+d} \to \mathbb{R} \mid h(t,\boldsymbol{x}) = \sum_{j=1}^{n} \beta_j \sigma\left(\alpha_j t + \sum_{i=1}^{d} w_{i,j} x_j + b_i\right) \right\} \tag{71}$$

such that $\check{C}(\sigma) = \cup_{n\geq 1}\check{C}^n(\sigma)$, σ is the activation function, and $(\alpha_j, w_{i,j}, b_i)$ are the parameters of the neural network. The PDE (66) approximated with $\hat{J}(t,\boldsymbol{x}|\Theta) \in \check{C}(\sigma)$ has the

[24] Note that by defining $\Omega_T := [T-(k-1)\Delta T, T-k\Delta T]\times\Omega$, $\Omega_C := (T-(k-1)\Delta t)\times\Omega_c$ and $\partial\Omega_T := (T-(k-1)\Delta t)\times\partial\Omega$, we recover the PDE (12) at kth time iteration. Also J_0 is mapped to $\tilde{J}$ in (12).

[25] In fact, $a(\cdot)$ and $b(\cdot)$ are treated as coefficients but I prove for a general case where they could be functions of the state variables.

total loss given by

$$\mathcal{L}(\hat{J}) = \left\|f(t,\boldsymbol{x})\right\|^2_{\Omega_T} + \left\|\frac{\partial \hat{J}(t,\boldsymbol{x}|\Theta)}{\partial \boldsymbol{x}}\right\|^2_{\partial\Omega_T} + \left\|\hat{J}(t,\boldsymbol{x}|\Theta) - J_0\right\|^2_{\Omega} + \tag{72}$$
$$\left\|\hat{J}(t,\boldsymbol{x}|\Theta) - J_0\right\|^2_{\Omega_c^1} + \left\|f(t,\boldsymbol{x})\right\|^2_{\Omega_c^2}$$
$$= \left\|f(t,\boldsymbol{x})\right\|^2_{\Omega_{C,T}} + \left\|\frac{\partial \hat{J}(t,\boldsymbol{x}|\Theta)}{\partial \boldsymbol{x}}\right\|^2_{\partial\Omega_T} + \left\|\hat{J}(t,\boldsymbol{x}|\Theta) - J_0\right\|^2_{\Omega_C}$$

where $\Omega_{C,T} := \Omega_c^2 \cup \Omega_T$ and $\Omega_C = \Omega \cup \Omega_c^1$

$$= \left\|\mathcal{G}[\hat{J}](t,\boldsymbol{x}|\Theta) - \mathcal{G}[J](t,\boldsymbol{x})\right\|^2_{\Omega_{C,T}} + \left\|\frac{\partial \hat{J}(t,\boldsymbol{x}|\Theta)}{\partial \boldsymbol{x}}\right\|^2_{\partial\Omega_T} + \left\|\hat{J}(t,\boldsymbol{x}|\Theta) - J_0\right\|^2_{\Omega_C}$$
$$\leq \int_{\Omega_{C,T}} \left|\frac{\partial J(\boldsymbol{x},t)}{\partial t} - \frac{\partial \hat{J}(t,\boldsymbol{x}|\Theta)}{\partial t}\right|^2 d\mu_1$$
$$+ \int_{\Omega_{C,T}} \left|\sum_{i=1}^{d} a(t,\boldsymbol{x})\left(\frac{\partial J(t,\boldsymbol{x})}{\partial x_i} - \frac{\partial \hat{J}(t,\boldsymbol{x})}{\partial x_i}\right)\right|^2 d\mu_1$$
$$+ \int_{\Omega_{C,T}} \left|\sum_{i,j=1}^{d} b(t,\boldsymbol{x})\left(\frac{\partial^2 J(t,\boldsymbol{x})}{\partial x_i x_j} - \frac{\partial^2 \hat{J}(t,\boldsymbol{x})}{\partial x_i x_j}\right)\right|^2 d\mu_1$$
$$+ \int_{\Omega_{C,T}} \left|\hat{A}(t,\boldsymbol{x},J(t,\boldsymbol{x}),\nabla J(t,\boldsymbol{x})) - \hat{A}(t,\boldsymbol{x},\hat{J}(t,\boldsymbol{x}),\nabla \hat{J}(t,\boldsymbol{x}))\right|^2 d\mu_1$$
$$+ \int_{\Omega_C} \left|\hat{J}(t,\boldsymbol{x}|\Theta) - J_0(t,\boldsymbol{x})\right|^2 d\mu_2 + \int_{\partial\Omega_T} \left|\frac{\partial \hat{J}(t,\boldsymbol{x}|\Theta)}{\partial \boldsymbol{x}}\right| d\mu_3 \tag{73}$$

Definition A.1. A subset $S \subset C^m(\Omega)$ is uniformly m-dense on compacts of $C^m(\Omega)$ if $\forall f \in C^m(\Omega)$, for all compact subsets X of Ω and $\forall \epsilon > 0$, $\exists g(f, X, \epsilon) \in S$ such that

$$||f - g||_{m,X} < \epsilon$$

Lemma A.1. *There exists a function $\hat{J} \in \mathcal{C}^n(\sigma)$ defined in (71) such that for all $\epsilon > 0$, we have*

$$\sup_{(t,x)\in\Omega_T} |J(t,\boldsymbol{x}) - \hat{J}(t,\boldsymbol{x}|\Theta)| + \sup_{(t,x)\in\Omega_T} \left| \frac{\partial J(t,\boldsymbol{x})}{\partial t} - \frac{\partial \hat{J}(t,\boldsymbol{x}|\Theta)}{\partial t} \right| + \max_{|i|\leq 2} \sup_{(t,\boldsymbol{x})\in\Omega_T} \left| \frac{\partial^{(i)} J(t,\boldsymbol{x})}{\partial \boldsymbol{x}^{(i)}} - \frac{\partial^{(i)} \hat{J}(t,\boldsymbol{x}|\Theta)}{\partial \boldsymbol{x}^i} \right| < \epsilon \tag{74}$$

Proof: This lemma is a direct consequence of Theorem 3 in Hornik [1991] with $m = 2$. The theorem states that if $\sigma \in C^m(\mathbb{R}^{1+d})$ is a non-constant and bounded function, then $\check{C}(\sigma)$ is uniformly m-dense on compacts in $C^m(\mathbb{R}^{1+d})$ ■

I restate (3.1) more formally before presenting the proof.

Theorem A.2. *Let $\check{C}(\sigma)$ be given by (71) with $\sigma(\cdot)$ a non-constant, bounded function, and let (μ_1, μ_2, μ_3) be the measures with support in $(\Omega_{C,T}, \Omega_C, \partial\Omega_T)$ respectively. Assume that $\hat{A}(t,x,y,z)$ is Lipschitz continuous with Lipschitz constants growing at most polynomially in y and z. Then, $\forall \epsilon > 0$, $\exists K > 0$ such that there exists a function $\hat{J}(t,x|\Theta) \in \check{C}(\sigma)$ satisfying*

$$\mathcal{L}(\hat{J}) < K\epsilon^2$$

Proof:

If $\hat{A}(t,x,y,z)$ is Lipschitz continuous with Lipschitz constants growing at most polynomially in y and z, we have

$$\begin{aligned} &\left| \hat{A}(t,x,\hat{J}(t,\boldsymbol{x}|\Theta), \nabla\hat{J}(t,\boldsymbol{x}|\Theta)) - \hat{A}(t,x,J(t,\boldsymbol{x}), \nabla J(t,\boldsymbol{x})) \right| \\ &\quad \leq \left(\left| \hat{J}(t,\boldsymbol{x}|\Theta) \right|^{a_1/2} + \left| \nabla\hat{J}(t,\boldsymbol{x}|\Theta) \right|^{a_2/2} + |J(t,\boldsymbol{x})|^{a_3/2} + |\nabla J(t,\boldsymbol{x})|^{a_4/2} \right) \\ &\quad \times \left(\left| \hat{J}(t,\boldsymbol{x}|\Theta) - J(t,\boldsymbol{x}) \right| + \left| \nabla\hat{J}(t,\boldsymbol{x}|\Theta) - \nabla J(t,\boldsymbol{x}) \right| \right) \end{aligned} \tag{75}$$

for some constants $0 < \{a_i\}_{i=1}^4 < \infty$ This condition will be crucial in proving convergence

as shown below. Applying Young's inequality with exponents $p_1 = p_2 = 2$, we get

$$\begin{aligned}
&\int_{\Omega_T} \left|\hat{A}(t,\boldsymbol{x},\hat{J}(t,\boldsymbol{x}|\Theta),\nabla\hat{J}(t,\boldsymbol{x}|\Theta)) - \hat{A}(t,\boldsymbol{x},J(t,\boldsymbol{x}),\nabla J(t,\boldsymbol{x})\right|^2 d\mu_1 \\
&\qquad \le 2\int_{\Omega_T} \left(\left|\hat{J}(t,\boldsymbol{x}|\Theta)\right|^{a_1} + \left|\nabla\hat{J}(t,\boldsymbol{x}|\Theta)\right|^{a_2} + |J(t,\boldsymbol{x})|^{a_3} + |\nabla J(t,\boldsymbol{x})|^{a_4}\right) \\
&\qquad \times \left(\left|\hat{J}(t,\boldsymbol{x}|\Theta) - J(t,\boldsymbol{x})\right| + \left|\nabla\hat{J}(t,\boldsymbol{x}|\Theta) - \nabla J(t,\boldsymbol{x})\right|\right) d\mu_1
\end{aligned}$$

Applying Hölder inequality $\|fg\|_1 \le \|f\|_p \|g\|_q$ for some constants $p, q \in [1,\infty)$ with $\frac{1}{p} + \frac{1}{q} = 1$, we have

$$\begin{aligned}
&\int_{\Omega_{C,T}} \left|\hat{A}(t,\boldsymbol{x},\hat{J}(t,\boldsymbol{x}|\Theta),\nabla\hat{J}(t,\boldsymbol{x}|\Theta)) - \hat{A}(t,\boldsymbol{x},J(t,\boldsymbol{x}),\nabla J(t,\boldsymbol{x})\right|^2 d\mu_1 \\
&\le 2\left(\int_{\Omega_{C,T}} \left(\left|\hat{J}(t,\boldsymbol{x}|\Theta)\right|^{a_1} + \left|\nabla\hat{J}(t,\boldsymbol{x}|\Theta)\right|^{a_2} + |J(t,\boldsymbol{x})|^{a_3} + |\nabla J(t,\boldsymbol{x})|^{a_4}\right)^p d\mu_1\right)^{1/p} \\
&\quad \left(\int_{\Omega_{C,T}} \left(\left|\hat{J}(t,\boldsymbol{x}|\Theta) - J(t,\boldsymbol{x})\right|^2 + \left|\nabla\hat{J}(t,\boldsymbol{x}|\Theta) - \nabla J(t,\boldsymbol{x})\right|^2\right)^q d\mu_1\right)^{1/q} \\
&\le C_1\left(\int_{\Omega_{C,T}} \left(\left|\hat{J}(\Theta) - J(t,\boldsymbol{x})\right|^{a_1} + \left|\nabla\hat{J}(\Theta) - \nabla J(t,\boldsymbol{x})\right|^{a2} + |J(t,\boldsymbol{x})|^{a_1\wedge a_3} + |\nabla J(t,\boldsymbol{x})|^{a_2\wedge a_4)}\right)^p d\mu_1\right)^{1/p} \\
&\quad \times \left(\int_{\Omega_{C,T}} \left(\left|\hat{J}(t,\boldsymbol{x}|\Theta) - J(t,\boldsymbol{x})\right|^2 + \left|\nabla\hat{J}(t,\boldsymbol{x}|\Theta) - \nabla J(t,\boldsymbol{x})\right|^2\right)^q d\mu_1\right)^{1/q} \\
&\le C_1\left(\epsilon^{a_1} + \epsilon^{a_2} + \sup_{\Omega_{C,T}} |J|^{a_1\wedge a_3} + \sup_{\Omega_{C,T}} |\nabla J|^{a_2\wedge a_4}\right)\mu_1(\Omega_{C,T})^{1/p}\left(2\epsilon^2\mu_1(\Omega_{C,T})^{1/q}\right) \\
&\le C_2\epsilon^2
\end{aligned}$$

where $C_1, C_2 < \infty$ are some constants that may depend on ϵ, and $\hat{J}(t,\boldsymbol{x} \mid \Theta)$ is abbreviated as $\hat{J}(\Theta)$ for brevity in some places. The condition (74) is used in the second last inequality. By Fubini Theorem,

$$\int_{\Omega_{C,T}} \left| \sum_{i=1}^{d} a(t,\boldsymbol{x}) \left(\frac{\partial J(t,\boldsymbol{x})}{\partial x_i} - \frac{\partial \hat{J}(t,\boldsymbol{x})}{\partial x_i} \right) \right|^2 d\mu_1 \leq \sum_{i=1}^{d} \int_{\Omega_{C,T}} a(t,\boldsymbol{x})^2 \left| \left(\frac{\partial J(t,\boldsymbol{x})}{\partial x_i} - \frac{\partial \hat{J}(t,\boldsymbol{x})}{\partial x_i} \right) \right|^2 d\mu_1$$
$$\leq \sum_{j=1}^{d} \int_{\Omega_{C,T}} A^2 \epsilon^2 d\mu_1 = dA^2 \mu_1(\Omega_{C,T}) \epsilon^2$$

where the constant $A < \infty$ bounds the function $a(t,\boldsymbol{x})$. Similarly,

$$\int_{\Omega_{C,T}} \left| \sum_{i,j=1}^{d} b(t,\boldsymbol{x}) \left(\frac{\partial^2 J(t,\boldsymbol{x})}{\partial x_j x_i} - \frac{\partial^2 \hat{J}(t,\boldsymbol{x})}{\partial x_j x_i} \right) \right|^2 d\mu_1 \leq \sum_{i=1}^{d} \int_{\Omega_{C,T}} b(t,\boldsymbol{x})^2 \left| \left(\frac{\partial^2 J(t,\boldsymbol{x})}{\partial x_j x_i} - \frac{\partial^2 \hat{J}(t,\boldsymbol{x})}{\partial x_j x_i} \right) \right|^2 d\mu_1$$
$$\leq \sum_{j=1}^{d} \int_{\Omega_{C,T}} B^2 \epsilon^2 d\mu_1 = dB^2 \mu_1(\Omega_{C,T}) \epsilon^2$$

where the constant $B < \infty$ bounds the function $b(t,\boldsymbol{x})$. Finally, from the condition (74), we have

$$\int_{\Omega_{C,T}} \left| \frac{\partial \hat{J}(t,\boldsymbol{x}|\Theta)}{\partial t} - \frac{J(t,\boldsymbol{x})}{\partial t} \right|^2 d\mu_1 \leq \epsilon^2 \mu_1(\Omega_{C,T})$$
$$\int_{\Omega_C} \left| \hat{J}(t,\boldsymbol{x}|\Theta) - J_0(t,\boldsymbol{x}) \right|^2 d\mu_2 \leq \epsilon^2 \mu_2(\Omega_C)$$
$$\int_{\partial\Omega_T} \left| \frac{\partial \hat{J}(t,\boldsymbol{x}|\Theta)}{\partial \boldsymbol{x}} \right|^2 d\mu_3 \leq \epsilon^2 \mu_3(\partial\Omega_T)$$

Putting it together, we have for some constant K

$$\mathcal{L} = \epsilon^2 \big(C_2 + dA^2 \mu_1(\Omega_{C,T}) + dB^2 \mu_1(\Omega_{C,T}) + \mu_1(\Omega_{C,T}) + \mu_2(\Omega_C) + \mu_3(\partial\Omega_T) \big) \big) = K\epsilon^2 \quad (76)$$

■

A.2 Benchmark model

A.2.1 Model set up

The return on capital held by each type of agent is given by

$$dR_{j,t} = \frac{d(q_t k_{j,t})}{q_t k_{j,t}} + \frac{(a_j - \iota_{j,t})}{q_t} dt$$

where q_t is the price of each unit of capital that follows the process

$$\frac{dq_t}{q_t} = \mu_t^q dt + \sigma_t^q dZ_t$$

The terms μ_t^q, and σ_t^q are endogenously determined in equilibrium. Using this dynamics for the price, the return process can be written as

$$dR_{j,t} = \underbrace{\left(\frac{a_j - \iota_{j,t}}{q_t} + \Phi(\iota_{j,t}) - \delta + \mu_t^q + \sigma\sigma_t^q \right)}_{\mu_{j,t}^R} dt + (\sigma + \sigma_t^q) dZ_t \tag{77}$$

Experts and households trade the capital and the experts are allowed to issue some outside equity. However, they have a skin in the game constraint: i.e, they have to hold at least a fraction $\underline{\chi} \in [0,1]$ of equity in their balance sheet. In addition to the risky capital, the agents also trade a risk free asset that pays a return r_t. Since the markets are not complete, there is no unique stochastic discount factor (SDF). Let $\xi_{e,t}$ and $\xi_{h,t}$ denote the SDF of experts and households respectively. Then, the process for SDF is given as

$$\frac{d\xi_{j,t}}{\xi_{j,t}} = -r_t dt - \zeta_{j,t} dZ_t \tag{78}$$

where, $\zeta_{j,t}$ is the market price of risk. Since both agents invest in the risk-free asset, the drift of the SDF process is the same for all agents. The aggregate output in the economy is given by

$$y_t = A_t K_t$$

where $K_t = \int_{\mathbb{E}\cup\mathbb{H}} k_{j,t}dj$, and A_t is the aggregate dividend that satisfies

$$A_t = \int_{\mathbb{H}} a_h \frac{k_{j,t}}{K_t} dj + \int_{\mathbb{E}} a_e \frac{k_{j,t}}{K_t} dj$$

Let the capital share held by expert sector be denoted by

$$\psi_t := \frac{\int_{\mathbb{E}} k_{j,t}dj}{\int_{\mathbb{H}\cup\mathbb{E}} k_{j,t}dj}$$

Equilibrium: The agents optimize by maximising their respective utility functions, subject to the wealth constraints starting from some initial wealth $w_{j,0}$. They solve

$$\sup_{c_{j,t},k_{j,t}} \quad E_t\left[\int_t^\infty f(c_{j,s},U_{j,s})ds\right] \tag{79}$$

$$\text{s.t.}\frac{dw_{j,t}}{w_{j,t}} = (r_t - \frac{c_{j,t}}{w_{j,t}} + \frac{q_t k_{j,t}}{w_{j,t}}\big((\mu^R_{j,t} - r_t) - (1-\chi_{j,t})\zeta_{j',t}\big)dt + \sigma_{w_j,t}(\sigma + \sigma^q_t)dZ_t \qquad j \in \{e,h\}$$

where $\chi_{j,t}$ denotes the skin-in-the game constraint of agent j, and j' denotes the other type of agent. The aggregator $f(c_{j,s},U_{j,s})$ is given in equation (2). By borrowing in the risk free market at a rate r_t and investing in risky capital, the agents obtain the market price of risk $\zeta_{j,t}$ less the compensation to outside equity holders. Note that since the agents retain only the fraction $\chi_{j,t}$ of risk in their balance sheet, the diffusion terms in wealth equation are given by

$$\sigma_{w_e,t} = \frac{q_t k_{e,t}}{w_{e,t}}\chi_t \tag{80}$$

$$\sigma_{w_h,t} = \frac{q_t k_{h,t}}{w_{h,t}} + (1-\chi_t)\frac{q_t w_{e,t}}{w_{h,t}} \tag{81}$$

The households do not issue outside equity and therefore $\chi_{h,t} = 1$. For the simplicity of notation, I denote $\chi_{e,t}$ as χ_t henceforth. The asset pricing conditions for the experts and the households are given by[26]

$$\frac{\frac{a_e - \iota_t}{q_t} + \Phi(\iota_t) - \delta + \mu^q_t + \sigma\sigma_{q,t} - r_t}{\sigma + \sigma_{q,t}} = \chi_t\zeta_{e,t} + (1-\chi_t)\zeta_{h,t} \tag{82}$$

[26]This can be proved using the Martingale argument. See Appendix **??** for the proof.

$$\frac{\frac{a_h - \iota_t}{q_t} + \Phi(\iota_t) - \delta + + \mu_t^q + \sigma\sigma_t^a - r_t}{\sigma + \sigma_t^q} \leq \zeta_{h,t} \tag{83}$$

When the risk premium demanded by the experts is large, they will sell maximum allowed equity to the households. Since the households do not issue outside equity, their asset pricing condition is simpler. The equality holds in (83) if the households own some amount of capital ($\psi_t < 1$). Combining the asset pricing conditions, we have

$$\frac{a_e - a_h}{q_t} \geq \chi_t(\zeta_{e,t} - \zeta_{h,t}) \tag{84}$$

$$\min\{\chi_t - \underline{\chi}, \zeta_{e,t} - \zeta_{h,t}\} = 0 \tag{85}$$

The equality in (84) holds when both the experts and the households hold capital. In this region, the experts issue maximum allowed equity ($\chi_t = \underline{\chi}$) as dictated by the condition (85). This is typically when the wealth share is low and the economy is in crisis state. The second region is when the premium of experts is still higher than that of the households but all capital is held by the experts and the economy is out of the crisis state. Since the premium of experts is higher, the experts issue maximum equity in this region as well. In the third region, there is perfect risk sharing where the premium of both type of agent becomes equal and χ_t is chosen to be equal to the wealth share z_t.

Solving the model: There are in fact an infinite number of agents in the economy but each individual in type $\mathbb{E}$ and $\mathbb{H}$ are identical and have the same preferences. Therefore, one can seek an equilibrium in which all agents in the same group take the same policy decisions. The system can be summarized with only one state variable: the wealth share of the experts which is sufficient to characterize the wealth distribution agents. It is defined as

$$z_t := \frac{W_{e,t}}{q_t k_t} \qquad \in (0,1)$$

where $W_{e,t} = \int_{\mathbb{E}} w_{j,t} dj$ and $K_t = \int_{\mathbb{E}} k_{j,t} dj + \int_{\mathbb{H}} k_{j,t} dj$. Moving forward, we write $X_{h,t}$ and $X_{e,t}$ to denote the aggregated quantity $\int_{\mathbb{H}} x_{j,t} dj$ and $\int_{\mathbb{E}} x_{j,t} dj$ respectively.

Proposition 5. *The law of motion of the wealth share of experts is given by*

$$\frac{dz_t}{z_t} = \mu_t^z dt + \sigma_t^z dZ_t \tag{86}$$

where

$$\mu_t^z = \frac{a_e - \iota_t}{q_t} - \frac{C_{e,t}}{W_{e,t}} + (\frac{\chi_t \psi_t}{z_t} - 1)(\sigma + \sigma_{q,t})(\zeta_{e,t} - (\sigma + \sigma_t^q)) + (1 - \chi_t)(\sigma + \sigma_t^q)(\zeta_{e,t} - \zeta_{h,t})$$
$$\sigma_t^z = (\frac{\chi_t \psi_t}{z_t} - 1)(\sigma + \sigma_t^q)$$

Proof: The law of motion of agents aggregated by their type is given by

$$\frac{dW_{h,t}}{W_{h,t}} = \Big(r_t - \frac{C_{h,t}}{W_{h,t}} + \frac{1 - \chi_t \psi_t}{1 - z_t}(\mu_{h,t}^R - r_t)\Big)dt + \frac{1 - \chi_t \psi_t}{1 - z_t}(\sigma + \sigma_t^q)dZ_t^k$$
$$\frac{dW_{e,t}}{W_{e,t}} = \Big(r_t - \frac{C_{e,t}}{W_{e,t}} + \frac{\chi_t \psi_t}{z_t}(\mu_{e,t}^R - r_t)\Big)dt + \frac{\chi_t \psi_t}{z_t}(\sigma + \sigma_t^q)dZ_t^k$$

where $W_{h,t} = \int_{j \in \mathbb{H}} w_{j,t} dj$ and similarly for the experts, and the expressions $\frac{q_t K_{e,t}}{W_{e,t}} = \frac{\psi_t}{z_t}$ and $\frac{q_t K_{h,t}}{W_{h,t}} = \frac{1 - \psi_t}{1 - z_t}$ are used along with the definition of z_t. ■

A.3 Brunnermeier-Sannikov meets Bansal-Yaron

A.3.1 Proof of asset pricing conditions

Consider the problem of experts first. The expected return earned from investing in the risky capital is given by

$$dr_t^v = (\mu_{e,t}^R - (1 - \chi_t)\bar{\epsilon}_{h,t})dt + \chi_t(\boldsymbol{\sigma_t^R})^T \boldsymbol{dZ_t}$$

where χ_t is the experts' inside equity share. The experts have to pay the outside equity holders $(1 - \chi_t)\bar{\epsilon}_{h,t}$ from the expected return that they earn, and hence this part is netted out from the drift. Since they are only exposed to a fraction χ_t of their total investment in the risky capital, the diffusion terms are multiplied by this fraction. For an investment of \$1 in the risky capital, the value of the investment strategy is given by

$$\frac{d(\xi_{e,t} v_t)}{\xi_{e,t} v_t} = (-r_t + \mu_{e,t}^R - (1 - \chi_t)\bar{\epsilon}_{h,t} - \chi_t \bar{\epsilon}_{e,t})dt + \text{difusion terms}$$

where the $\bar{\epsilon}_{e,t} = \boldsymbol{\zeta}_{e,t}^T \boldsymbol{\sigma}_t^{\boldsymbol{R}}$ and $\xi_{e,t}$ is the SDF process given by (78). Since the stochastic process v_t is a martingale, the drift term should be zero. This gives us

$$\mu_{e,t}^R - t_t = \chi_t \bar{\epsilon}_{e,t} + (1 - \chi_t)\bar{\epsilon}_{h,t}$$

The asset pricing condition for the households follows in a similar fashion except that they do not issue outside equity, and hence we arrive at

$$\mu_{h,t}^R - r_t = \bar{\epsilon}_{h,t}$$

where $\bar{\epsilon}_{h,t} = \boldsymbol{\zeta}_{h,t}^T \boldsymbol{\sigma}_t^{\boldsymbol{R}}$ ■

A.3.2 Proof of Proposition 1

The law of motion of the wealth obtained by aggregating wealth and using law of large numbers is given by

$$\frac{dW_{h,t}}{W_{h,t}} = \Big(r_t - \hat{C}_{h,t} + \theta_{h,t}(\mu_{h,t}^R - r_t) + \tau_t \frac{W_{e,t}}{W_{h,t}}\Big)dt + \theta_{h,t}(\boldsymbol{\sigma}_{\boldsymbol{t}}^{\boldsymbol{R}})^T \boldsymbol{dZ_t} \tag{87}$$

$$\frac{dW_{e,t}}{W_{e,t}} = \big(r_t - \hat{C}_{e,t} + \theta_{e,t}\bar{\epsilon}_{e,t} - \tau_t\big)dt + \theta_{e,t}(\boldsymbol{\sigma}_{\boldsymbol{t}}^{\boldsymbol{R}})^T \boldsymbol{dZ_t} \tag{88}$$

where $W_{h,t} = \int_{\mathbb{H}} w_{j,t} dj$ and $W_{e,t} = \int_{\mathbb{E}} w_{j,t} dj$ are the aggregate wealth within the respective class. The wealth share of the experts is defined to be $z_t = \dfrac{W_{e,t}}{W_{e,t} + W_{h,t}}$. Also, the variables $\theta_{e,t} := \dfrac{\chi_t \psi_t}{z_t}$ and by market clearing, $\theta_{h,t} = \dfrac{1 - \chi_t \psi_t}{1 - z_t}$

Applying Ito's lemma, we get

$$\frac{dz_t}{z_t} = \frac{dW_{e,t}}{W_{e,t}} - \frac{d(q_t k_t)}{q_t k_t} + \frac{d\langle q_t k_t, q_t k_t\rangle}{(q_t k_t)^2} - \frac{d\langle q_t k_t, W_{e,t}\rangle}{(q_t k_t W_{e,t})}$$

where

$$\frac{d(q_t k_t)}{q_t k_t} = (\bar{\epsilon}_{e,t}(\sigma + \sigma_t^q) - \frac{(a_{e,t} - \iota_t)}{q_t} + r_t)dt + (\boldsymbol{\sigma_t^R})^T \boldsymbol{dZ_t}$$
$$\frac{d\langle q_t k_t, q_t k_t \rangle}{(q_t k_t)^2} = ||\boldsymbol{\sigma_t^R}||^2 dt$$
$$\frac{d\langle q_t k_t, w_{e,t} \rangle}{q_t k_t w_{e,t}} = \left(\theta_{e,t} ||\boldsymbol{\sigma_t^R}||^2\right) dt$$

We get the desired result after few steps of algebra. ■

A.3.3 Proof of Proposition 2

The conjecture for the value function is

$$U_{j,t} = \frac{(J_{j,t}(\boldsymbol{x}) K_t)^{1-\gamma}}{1-\gamma} \tag{89}$$

where K_t is the aggregate capital, and the the stochastic opportunity set $J_{j,t}$ satisfies the equation

$$\frac{dJ_{j,t}}{J_{j,t}} = \mu_{j,t}^J dt + (\boldsymbol{\sigma_t^J})^T \boldsymbol{dZ_t}$$

The objects $\mu_{j,t}^J$ and $\boldsymbol{\sigma_t^R}$ needs to be determined in equilibrium. Applying Ito's lemma to $U_{j,t}$ and using the HJB equation $\sup_{c,k} f(c_{j,t}, U_{j,t}) + E[dU_{j,t}] = 0$, with

$$f(c_{j,t}, U_{j,t}) = (1-\gamma)\rho U_{j,t}\left(\log c_{j,t} - \frac{1}{1-\gamma}\log\left((1-\gamma)U_{j,t}\right)\right)$$

we get

$$\begin{aligned}\sup_C \quad & \rho(J_{j,t}K_t)^{1-\gamma}[\log\frac{C_{j,t}}{W_{j,t}} - \log J_{j,t} + \log(q_t z_{j,t})] + \frac{(J_{j,t}K_t)^{1-\gamma}}{1-\gamma}\mu_{j,t}^J + (J_{j,t}K_t)^{1-\gamma}(\Phi(\iota_t - \delta)) \\ & - (J_{j,t}K_t)^{1-\gamma}\frac{1}{2}\gamma\sigma^2 + (1-\gamma)(J_{j,t}K_t)^{1-\gamma}\sigma\sigma_{j,t}^{J,k} - \frac{\gamma}{2}(J_{j,t}K_t)^{1-\gamma}||\sigma^J||^2 + \tau_t(U_{h,t} - U_{e,t}) = 0\end{aligned} \tag{90}$$

By envelope condition, the marginal utilities of wealth and consumption should equal at the optimum. Since $\tilde{J} = \frac{J}{qz}$, we can rewrite

$$U_{j,t} = \frac{(\tilde{J}_{j,t} W_{j,t})^{1-\gamma}}{1-\gamma}; \qquad f(C_{j,t}, U_{j,t}) = (1-\gamma)\rho U_{j,t}(log\frac{C_{j,t}}{W_{j,t}} - \tilde{J}_{j,t}) \tag{91}$$

Using this, we have

$$\frac{\partial U_{j,t}}{\partial W_{j,t}} = \frac{\partial f_{j,t}}{\partial C_{j,t}} \implies (\tilde{J}_{j,t} W_{j,t})^{-\gamma} = (1-\gamma)\rho \frac{U_{j,t}}{C_{j,t}} \implies \frac{C_{j,t}}{W_{j,t}} = \rho$$

That is, the optimal consumption-wealth ratio is equal to the discount rate. The SDF for recursive utility is expression as

$$\xi_{j,t} = exp\left(\int_0^t \frac{\partial f(C_{j,s}, U_{j,s})}{\partial U} ds\right) \frac{\partial U_{j,t}}{\partial W_{j,t}}$$

Utilizing (91), we get

$$\xi_{j,t} = exp\left(\int_0^t \left[(1-\gamma)\rho(\log\rho - \tilde{J}_{j,t})\right] ds\right) \frac{U_{j,t}}{W_{j,t}}$$

Thus, the volatility of the SDF is equal to the volatility of the quantity $\frac{U}{W}$. Let us define $v(J, \boldsymbol{x}) := \frac{U}{W}$. Using Ito's lemma, equating the coefficients of volatility terms to the volatility of SDF equation (78), we get the result. ■

A.3.4 Proof of Proposition 3

The first equation (59) comes from plugging in the risk premium from (2) in the asset pricing condition (84). The volatility of the opportunity set σ^J in the risk premium is expressed in terms of the partial derivative of J with respect to the state variables. This can be easily derived using Ito's lemma and comparing the diffusion terms of (31) and $dJ(z, \boldsymbol{x})$. The second equation (60) comes from the capital market clearing condition and using $z_t = \frac{W_t}{q_t K_t}$, $\psi_t = \frac{K_{e,t}}{q_t K_t}$. To derive (61) and (62), first apply Ito's lemma to $q(z_t, \boldsymbol{x})$ to

get

$$\begin{aligned} dq(\boldsymbol{x}) &= \frac{\partial q}{\partial \boldsymbol{x}} \boldsymbol{dx} + \frac{1}{2} \frac{\partial^2 q}{\partial^2 \boldsymbol{x}^2} d\langle \boldsymbol{x}, \boldsymbol{x} \rangle \\ &= \text{drift terms} + \frac{\partial q}{\partial \boldsymbol{x}} \boldsymbol{\sigma^x} \end{aligned}$$

Matching the volatility terms with the capital price equation (4), we get the result. ■

A.3.5 Proof of Proposition 4

Applying Ito's lemma to $J(t, \boldsymbol{x})$, we get

$$\begin{aligned} dJ(\boldsymbol{x}) &= \frac{\partial J}{\partial \boldsymbol{x}} d\boldsymbol{x} + \frac{1}{2} \frac{\partial^2 J}{\partial \boldsymbol{x}^2} d\langle \boldsymbol{x}, \boldsymbol{x} \rangle) dt \\ &= \left(\frac{\partial J}{\partial \boldsymbol{x}} \boldsymbol{\mu^x} + \frac{1}{2} \frac{\partial^2 J}{\partial \boldsymbol{x}^2} (\boldsymbol{\sigma^x})^2 \right) dt + \text{volatility terms} \end{aligned}$$

Comparing this with the equation (31) and matching the drift terms, we get the expression

$$J\mu^J = \frac{\partial J}{\partial \boldsymbol{x}} \boldsymbol{\mu^x} + \frac{1}{2} \frac{\partial^2 J}{\partial \boldsymbol{x}^2} (\boldsymbol{\sigma^x})^2$$

Adding the fase time-derivative, it remains to derive the quantity μ^J which can be obtained from the HJB equation (90). The term $A\left(\boldsymbol{x}, J, \frac{\partial J}{\partial \boldsymbol{x}}\right)$ includes both μ^J, as well as the diffusion term $\frac{\partial J}{\partial \boldsymbol{x}} \boldsymbol{\mu^x}$. The term $B\left(\boldsymbol{x}, J, \frac{\partial J}{\partial \boldsymbol{x}}\right)$ represents the diffusion terms $(\boldsymbol{\sigma^x})^2$. This proves the proposition. ■

A.3.6 Three-dimensional plots

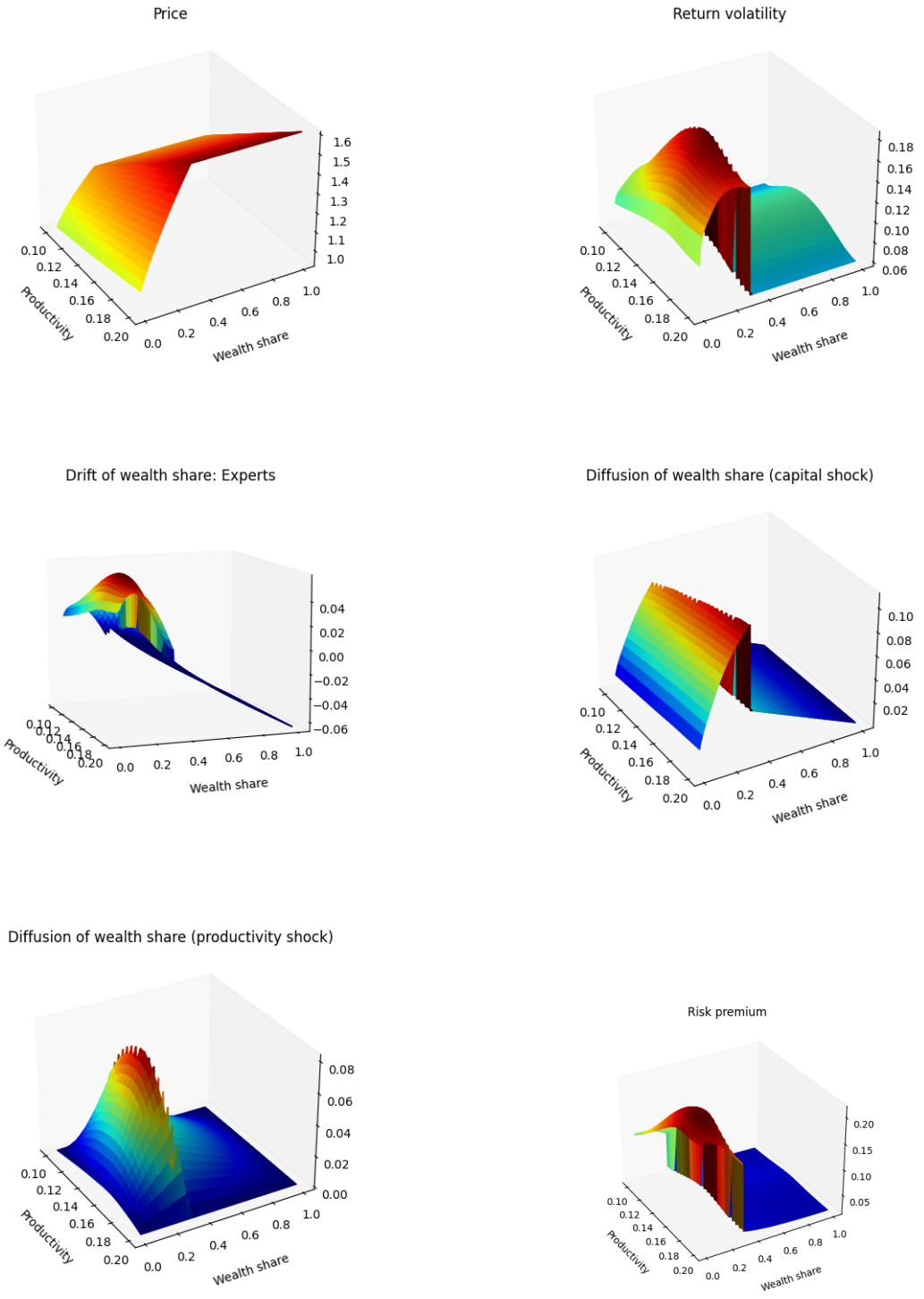

Figure 7. Equilibrium quantities as a function of state variables z_t and s_t. Growth rate (g_t) and productivity ($a_{e,t}$) are fixed at respective average values.

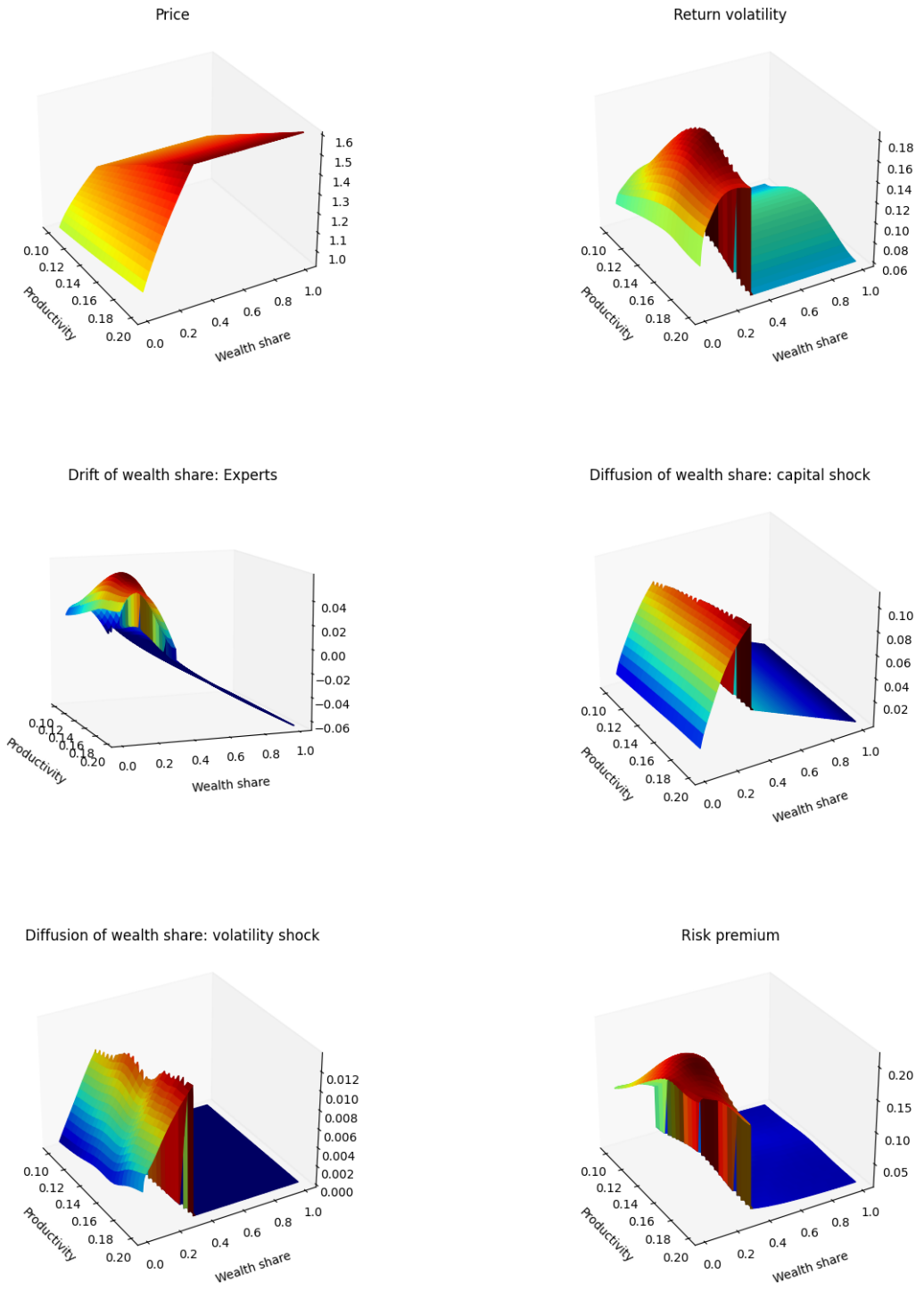

Figure 8. Equilibrium quantities as a function of state variables z_t and $a_{e,t}$. Growth rate (g_t) and volatility (s_t) are fixed at respective average values.

A.3.7 Implementation details

Data efficiency: One of the main advantages of using neural network to fit the PDE is data efficiency. For the benchmark model, I use 1000 grid points for space dimension in the inner Newton-Raphson method. When it comes to training the neural network, I randomly sample 300 points in each iteration. At kth iteration, the function to be solved is denoted by $J(T - k\Delta t, z)$ whose economic behavior is governed by the given PDE in the domain $[T - k\Delta t, T - (k-1)\Delta t] \times \Omega_z$. I randomly sample time points from the range $[T - k\Delta t, T - (k-1)\Delta t]$ in order to avoid errors in computing gradients with respect to time dimension. The 300 grid points include these time points as well. Figure () presents the sparse grid used for training. The task is to solve for the function $J(T - k\Delta t, z)$ such that the PDE is respected in the domain $[T - k\Delta t, T - (k-1)\Delta t]$ along with bounding conditions in the domain $(T - (k-1)\Delta t) \times \Omega$ and $(T - (k-1)\Delta t) \times \partial\Omega$.

Simplicity in coding: I rely on Tensorflow, a popular library developed by Google to copmute derivatives in an efficient way. The method $tf.gradients$ computes the required symbolic partial derivatives, which are then collected to form the PDE loss residual. In a similar fashion, the bounding network and active network can computed. An important thing to note is that the module $tf.gradients$ creates a computational graph and does not perform the calculation yet. Once all networks in ALIEN are built, one can start the tensoflow session which then initiates the computation of gradients. This allows us to build the model first and then distribute the data efficiently as demonstrated later. The code snipper (1) shows how to approximate the function J using a neural network. The inputs are the state variables along with the weights and biases which are the parameters of the neural network. Before training begins, the weights are initialized using Xavier initialization as explained earlier. The snippet (2) demonstrates the computation of PDE network. The inputs are the function $\hat{J}$ approximated using the neural network along with advection, diffusion, and linear terms which are PDE coefficients.[27] The gradients of the approximated function $\hat{J}$ with respect to the state variables are computed using automatic differentiation through the tensorflow module $tf.gradients$. Note that automatic differentiation is commonly used in machine learning to obtain derivatives of functions with respect to the neural network parameters.

[27] The advection, diffusion, and linear term coefficients are calculated before starting the training process and hence they can be simply passed as inputs into the neural network algorithm.

Here, I utilize automatic differentiation to obtain derivatives with respect to the state variables. Apart from this difference, the gradient computation is standard. One can notice that the coding effort involved in computing the derivatives is very minimal.

```
def J(z,t):
  J = neural_net(tf.concat([z,t],1),weights,biases)
  return J

```

Listing 1: Approximating J using a neural network: Benchmark model

```
def f(z,t):
  #compute fundamental network Jhat
  J = J(z,t)
  #compute first partial derivatives
  J_t = tf.gradients(J,t)[0]
  J_z = tf.gradients(J,z)[0]
  #compute second partial derivatibes
  J_zz = tf.gradients(J_z,z)[0]
  #compute PDE residual
  f = J_t + advection * J_z + diffusion * J_zz - linearTerm * J
  return f
```

Listing 2: Constructing a regularizer: Benchmark model

While the benchmark model can be solved using traditional methods discussed earlier, it is not trivial to extend these methods to solve models in higher dimensions. In finite difference schemes, it is not only problematic to maintain the monotonicity but also difficult to code especially in the case of implicit schemes. For example, Hansen et al. [2018] solves a collection of nested macro-finance models in 3 dimensions which involves setting up dimension-specific matrices to solve the PDEs. It is not only difficult to code but also requires high performance computing libraries such as Paradiso (specific to C++) to solve large linear systems that show up in the implicit finite difference scheme. In contract, the framework proposed in this paper involves less coding effort in scaling to higher dimensions. To appreciate the simplicity, I demonstrate sample codes from the capital misallocation model considered in Section 5. In code snippet (3), the function J is approximated using the neural network. The inputs are 4 state variables and 1 time dimension, along with weights and biases which are the parameters of the network. As before, the weights go through Xavier initialization before the learning begins. The snippet (4) constructs the regularizer corresponding to the PDE network. The inputs are approximated function $\hat{J}$ along with the PDE coefficients that are known. Using automatic differentiation, one can easily obtain the partial derivatives and compute the PDE residual. It only takes a few additional lines of coding to move from a 1 dimensional model to 4 dimensional model. In contrast, it is not at all trivial to move easily to higher dimensions using finite difference schemes. The ease in implementation

shifts the burden from the modeler to Tensorflow thereby freeing up time to focus on more important issues from an economic standpoint.

```
def J(z,t):
  J = neural_net(tf.concat([z,t],1),weights,biases)
  return J

```

Listing 3: Approximating J using a neural network: 4D model

```
def net_f(z,g,s,a,t):
        #compute fundamental network Jhat
        J = J(z,g,s,a,t)
        #compute first partial derivatives
        J_z, J_g = tf.gradients(J,z)[0], tf.gradients(J,g)[0]
        J_s, J_a = tf.gradients(J,s)[0], tf.gradients(J,a)[0]
        J_t = tf.gradients(J,t)[0]
        #compute second partial derivatives
        J_zz = tf.gradients(J_z,z)[0]
        J_gg = tf.gradients(J_g,g)[0]
        J_ss = tf.gradients(J_s,s)[0]
        J_aa = tf.gradients(J_a,a)[0]
        J_zg = tf.gradients(J_z,g)[0]
        J_zs = tf.gradients(J_z,s)[0]
        J_za = tf.gradients(J_z,a)[0]
        #compute PDE residual
        f =  J_t + diffusion_z * J_zz +  diffusion_g * J_gg + \
                   diffusion_s * J_ss + diffusion_a * J_aa + \
                   advection_z * J_z + advection_g * J_g + \
                   advection_s * J_s + advection_a * J_a + \
                   cross_term_zg * J_zg + cross_term_zs * J_zs + \
                   cross_term_za * J_za -  linearTerm * J
        return f
```

Listing 4: Constructing a regularizer: 4D model

Distributed learning: The concept of distributed learning is not new and entails utilization of multiple workers or GPUs to speed up computation. Specifically, data parallelism works by dividing up the data into pieces (or mini-batches in the language of machine learning) and sending to different workers that will run the data through the same model. Algorithm 2 presents a simple data parallelism approach. There are a few bottlenecks presented by this procedure. First, it requires the user to employ a library that can communicate across workers. Secondly, and more importantly, the communication overhead resulting from the cross-worker interface may be significant thereby defeating the purpose of using a distributed algorithm. For example, Sergeev and Balso [2018a] finds that roughly half of the computational resources are lost due to the overhead when they train a big data model on 128 GPUs. The reason for such heavy overhead is that the default way of communicating across workers is through a parameter sharing approach, where each node assumes the role of either a worker or a parameter server. The role of worker is to train the model, and the parameter server aggregates the gradients. The user is left to decide the optimal ratio of parameter server to worker. A small ratio leads to a large computational bottleneck, and a large ratio leads to

communication overhead. Andrew Gibiansky [2017] at Baidu presents an inter-worker communication algorithm that bypasses these problems. It is based on ring-AllReduce, where each worker communicates with its two neighbours only in a ring-like fashion for a total of $2 * (N - 1)$ times. During the first $N - 1$ communications, each worker sends its data to the neighbours, and receives the data from the neighbors to store it in the buffer. In the next $N - 1$ communications, the workers receive the data from the neighbours and update the buffer. This algorithm is shown to the optimal one for the utilization of the bandwidth provided the buffer is large enough.

Algorithm 2

```
1:  procedure DISTRIBUTED ALGORITHM
2:      Assign a chief worker
3:      Divide data by number of workers
4:      while worker < N do                          ▷ N is number of workers
5:          Assign model to current worker
6:          Run the data through the model
7:          Compute gradients
8:          Send gradients to chief worker
9:      Average gradients from multiple workers
10:     Update the model
```

Horovod: Sergeev and Balso [2018a] leverages the advantages of ring-AllReduce algorithm and combines it with Tensorflow to build libraries that facilities easy implementation of distributed learning. The user only has to add a few lines to code to enable a hybrid parallelization of the deep learning algorithm. Algorithm 3 presents the pseudo-code for implementing Horovod.

Algorithm 3

```
1:  procedure HOROVOD
2:      Initialize Horovod
3:      Assign a GPU to each tensorflow process
4:      Start a tensorflow session
5:      Split data based on number of workers
6:      Build deep learning model and set up loss functions
7:      Wrap the optimizer with Horovod optimizer        ▷ To average gradients
8:      Initiate Tensorflow session to train the deep learning model
9:      Broadcast variables from chief worker to all other workers   ▷ This makes sure
    that all workers have the same initial parameters in the model.
10:     Train the model until convergence
```

The code snippet (5) presents the implementation of Horovod to ALIENs. Each line in the pseudo-code (3) can be implemented simply by calling a module in Horovod library as seen in the code snippet. This again shifts the burden from the modeler to the library that not only frees up the modeler's time but also eliminates the necessity to deal with inter-worker communication issues that are rarely of interest to an economist.

```
def J():
  ...
def net_f():
  ...

hvd.init() #initialize Horovod
config = tf.ConfigProto() #pin GPUs to processes
config.gpu_options.visible_device_list = str(hvd.local_rank()) #assign chief worker
config.gpu_options.allow_growth = True #enable GPU
sess= tf.Session(config=config) #Configure tensorflow
if hvd.rank()==0:
  ... #assign a piece of data to chief worker
else:
  while hvd.rank() < hvd.size():
    ... #assign a piece of data to each worker

def build_model():
  #initialize parameters using Xavier initialization
  #parametrize the function J using J()
  #buld loss function using net_f()
  #set up tensorflow optimizer in the variable name opt
  optimizer = hvd.DistributedOptimizer(opt)
  #minimize loss
  #initialize Tensorflow session
  bcast = hvd.broadcast_global_variables(0) #Broadcast parameters to all workers
  sess.run(bcast)
  #train the deep learning model
```

Listing 5: ALIENs using Horovod