Deep Learning Solutions to Master Equations for Continuous Time Heterogeneous Agent Macroeconomic Models

Zhouzhou Gu[*], Mathieu Laurière[†], Sebastian Merkel[‡], Jonathan Payne[§¶]

April 29, 2024

Abstract

We propose and compare new global solution algorithms for continuous time heterogeneous agent economies with aggregate shocks. First, we approximate the state space so that equilibrium in the economy can be characterized by one high, but finite, dimensional partial differential equation. We consider different approximations: discretizing the number of agents, discretizing the state variables, and projecting the distribution onto a set of basis functions. Second, we approximate the value function using neural networks and solve the differential equation using deep learning tools. We refer to the solution as an Economic Model Informed Neural Network (EMINN). The main advantage of this technique is that it allows us to find global solutions to high dimensional, non-linear problems. We demonstrate our algorithm by solving canonical models in the macroeconomics literature.

Keywords: Heterogeneous agents, computational methods, deep learning, inequality, mean field games, continuous time methods, aggregate shocks, global solution.

[*]Princeton, Department of Economics. Email: zg3990@princeton.edu

[†]Shanghai Frontiers Science Center of Artificial Intelligence and Deep Learning; NYU-ECNU Institute of Mathematical Sciences at NYU Shanghai. Email: mathieu.lauriere@nyu.edu

[‡]University of Exeter, Department of Economics. Email: s.merkel@exeter.ac.uk

[§]Princeton, Department of Economics. Email: jepayne@princeton.edu

[¶]We would like to thank Adam Rebei for outstanding research assistance throughout this project. We are also very grateful to the comments and discussion from Mark Aguiar, Yacine Ait-Sahalia, Marlon Azinovic, Johannes Brumm, Markus Brunnermeier, Rene Carmona, Wouter Den Haan, Goutham Gopalakrishna, Ji Huang, Felix Kubler, Moritz Lenel, Ben Moll, Galo Nuno, Thomas J. Sargent, Simon Scheidegger, Jesus Fernandez Villaverde, Yucheng Yang, and Jan Žemlička. We are also thankful for the comments from participants in the DSE 2023 Conference on Deep Learning for Solving and Estimating Dynamic Models, the CEF 2023 Conference, the IMSI 2023 Conference, the 11th Western Conference on Mathematical Finance, the Macro-finance Reading Group at USC Marshall, and the Princeton Civitas Seminar. We thank Princeton Research Computing Cluster for their essential computational resources.

1 Introduction

Macroeconomists have great interest in studying models with heterogeneous agents and aggregate shocks. However, a major difficulty with working on these models is that the agent distribution of individual states becomes an aggregate state variable and so the state space becomes infinite dimensional. In recent years, we have seen many attempts to address this problem using tools from the deep learning literature. All approaches need to start by constructing a finite dimensional approximation to the distribution of agent positions. Most approaches are in discrete time and replace the agent continuum by a finite collection of agents. In this paper, we develop and compare the three main approaches for approximating the distribution in continuous time: imposing a finite number of agents, discretizing the state space, and projecting onto a collection of basis functions. For each approximation, we show how to characterize general equilibrium as a high but finite dimensional differential equation and how to customize deep learning techniques to compute global numerical solutions to the differential equation.

We develop solution techniques for a class of dynamic, stochastic, general equilibrium economic models with the following features. There is a large collection of price-taking agents who face uninsurable idiosyncratic and aggregate shocks. Given their belief about the evolution of aggregate state variables, agents choose control processes to solve dynamic optimization problems. When making their decisions, agents face financial frictions that constrain their behaviour and potentially break "aggregation" results that would allow the distribution of agents to be replaced by a "representative agent". We solve for a rational expectations equilibrium in which agent beliefs about the evolution of aggregate states are consistent with the dynamics that emerge in the economy. Solving for equilibrium reduces to solving a "master" partial differential equation (PDE) that summarizes both the agent optimization behaviour (from the Hamilton-Jacobi-Bellman equation) and the evolution of the distribution (from the Kolmogorov forward equation). A canonical example of this type of environment is the Krusell and Smith (1998) model, which is often used as a workhorse environment for testing solution methods in macroeconomics and we ultimately use as our main example economic model.

Our solution approach approximates the infinite dimensional master equation by a finite, but high, dimensional PDE and then uses deep learning to solve the high dimensional equation. We consider three different approaches for reducing the dimension of the master equation. The first approach approximates the distribution by a large, finite number of agents. We refer to this as the "finite-agent" approximation. The second approach approximates the distribution by discretizing the agent state space so the density becomes a collection of mass points at grid points. We call this the "discrete-state" approximation. The third approach approximates the distribution by a linear combination of finitely many basis functions. We call this the "projection" approximation. Most deep learning macroeconomic papers have focused on the finite agent approximation with some exceptions that use a discrete-state approximation (e.g. Huang (2022)).

We solve the finite dimensional approximation to the master equation using recent advances in deep learning. In particular, we adapt the Deep Galerkin Method (DGM) developed by the applied mathematics, which is similar to the Physics Informed Neural Networks (PINNs) developed in the physics literature. This approach approximates the value function by a neural network and then uses stochastic gradient descent to train the neural network to minimise a loss function that summarizes the average error in the master equation. We calculate average errors by randomly sampling over points in the state space, with greater sampling applied to the regions with more curvature. We also need to handle inequality boundary conditions arising from financial constraints on the evolution of the state space. We do this by introducing penalty functions for when financial constraints bind. We refer to our approach as training Economic Model Informed Neural Networks (EMINNs).

Although our deep learning approach is relatively simple to describe at an abstract level, there are many complicated implementation details that need to be worked through. In this sense neural network training is more like an "art" than a routine procedure. One implementation detail that is particularly important for our approach is choosing how to sample the states on which we evaluate the master differential equation equation loss. This is different to discrete time approaches where the economy is simulated to calculate expectations and is also different to other continuous time deep learning papers that don't have high dimensional distributions as states. We consider three approaches for sampling the distribution. The first is "moment sampling" that draws samples for selected moments of the distribution and then samples agent distributions that satisfy the selected moments. The second is "mixed steady state sampling" that solves for the steady state for a collection of fixed aggregate states and then samples random mixtures of the different steady state distributions. The third is ergodic sampling that samples dynamically by regularly simulating the model economy based on the current estimate of the model solution.

We have found different sampling strategies are useful for different types of distribution approximations. For the finite agent approximation, we found moment sampling to be simple and effective. For the discrete state space approximation, we find the most stable approach was to start with mixed steady state sampling and then move to ergodic sampling once the neural network started to converge. For the projection approximation, we find that a combination of moment sampling and ergodic sampling was effective. To put moment sampling to work with projections requires a rotation of the basis functions so as to isolate components that correspond to the selected moments. Deep learning techniques are sometimes referred to as "breaking the curse of dimensionality" but this is not really true in the sense that we can only show the neural network a very sparse set of distributions. Instead, a key "art" to training a neural network is to sample from an intelligently chosen subspace that helps the neural network to learn the equilibrium functional relationships in an economically interesting part of the state space.

We illustrate our techniques by solving continuous time versions of the Aiyagari (1994) and Krusell and Smith (1998) models (with some simple extensions in the appendix). These are canonical macroeconomic models that have well developed solution approaches that we can use to test our method. For the Aiyagari (1994) model, we show that we can match the finite difference solution to high accuracy and, for the finite-agent approximation, solve for transition dynamics without a shooting algorithm. For the Krusell and Smith (1998) model, we show that we generate solutions with a low error for the master equation. Traditional techniques only give approximate solutions to the Krusell and Smith (1998) model so we do not have an ideal benchmark for testing the model. However, we show that we get very similar results to contemporary approaches such as Fernández-Villaverde et al. (2023). Ultimately, all the distribution approximations are able to solve the Krusell and Smith (1998) model.

Although the different distribution approximation approaches can all be effective, we found they had different strengths and weaknesses in our experiments. We found the finite-agent approximation to be very robust in a number of ways: the neural network can be trained with a simple sampling procedure that does not require ergodic simulation (or any knowledge of the model solution), the algorithm only required a moderate number of agents (approximately 40), and we had success adding parameters as auxiliary states so the model could be solved across the state and parameter space at the same time. By contrast, we found the discrete-state approximation to be difficult to work with for the Krusell and Smith (1998) model. We believe a lot of the issues come from having to approximate the derivatives in the Kolmogorov Forward Equation on the discrete state space. One challenge is that this requires a fine grid for agent wealth (approximately 200 grid points). A related challenge is that the training samples need to come from relatively smooth densities and so ergodic sampling is very important. Ultimately, this made the model slow to train. Finally, the projection approach brings a different set of trade-offs. Both the finite agent and discrete state approximations feed relatively large state spaces into the neural network and then let the neural network work out how to structure the approximate value function. By contrast, the projection method requires more choices ex-ante. This allows us to work with a much lower dimensional approximation (approximately 5 basis functions) and to choose which statistics of the distribution we want to match most closely in our approximation.

Literature Review: The economics literature has traditionally used three main approaches for solving heterogeneous agent models with aggregate shocks. One approach is to fit a statistical approximation to the law of motion for the key aggregate state variables (e.g. Krusell and Smith (1998), Den Haan (1997), Fernández-Villaverde et al. (2023)). As has been extensively discussed in the literature, this approach works well when the law of motion for the key state variables can be efficiently approximated as a function of key moments of the distribution (and so the economy is very close to permitting "aggregation"). By contrast, our approach can handle economies without near-aggregation results. A second approach is to take a type of perturbation in the aggregate state and then solve the resulting problem with matrix algebra (e.g. Reiter (2002), Reiter (2008), Reiter (2010), Winberry (2018), Ahn et al. (2018), Auclert et al. (2021), Bilal (2021), Bhandari et al. (2023)). By contrast, we solve the model globally and so can handle partial differential equations with extensive non-linearity. A final approach is to take a low dimensional projection of the distribution (e.g. Prohl (2017), Schaab (2020)). Our approach is complementary to these papers in that it allows for more general, higher dimensional projections through the use of deep neural networks.

Our paper is part of a growing computational economics literature using deep learning techniques to solve economic models and overcome the limitations of the traditional solution techniques. Many of these papers focus on solving heterogeneous agent macroeconomic models in discrete time (e.g. Azinovic et al. (2022), Han et al. (2021), Maliar et al. (2021), Kahou et al. (2021), Bretscher et al. (2022)) or using a discrete time approximation to a system forward and backward differential stochastic equations (e.g. Han et al. (2018), Huang (2022)). Our work is part of a less developed literature attempting to deploy deep learning techniques to solve the differential equations that arise in continuous time economic models (e.g. Duarte (2018), Gopalakrishna (2021), Fernandez-Villaverde et al. (2020), Sauzet (2021)). We make two main contributions. First, within the continuous time literature, we show how to handle a rich distribution of agents and directly solve the "master equation" for the economic system. This type of PDE has been introduced by Lions (2011) to describe the value function of a representative player in a mean field population of players at equilibrium. This necessitates resolving a collection of problems that are particular to the analytic characterization of continuous time problems, such as: approximating inequality boundary conditions, using the Kolmogorov Forward Equation to derive laws of motion for projection coefficients, and approximating derivatives with respect to the distribution. Second, compared to all the discrete time literature, we compare a range of finite dimensional distribution approximations rather than imposing a finite agent approximation (or a finite state space approximation in Han et al. (2018)). We extend our technique outlined in this paper to solve search and matching models in Payne et al. (2024) and macro-finance models with long-term asset prices in Gopalakrishna et al. (2024).

There is also a growing mathematics literature that attempts to use neural networks to solve differential equations. Technically, our approach builds on the Deep Galerkin Method (DGM) and Physics Informed Neural Networks (PINNs) developed in Sirignano and Spiliopoulos (2018) and Raissi et al. (2017); see also Li et al. (2022). These papers train PINNs to solve systems of differential equations that arise in physics and finance for instance. We train Economic Model Informed Neural Networks (EMINNs) to solves systems of differential equations that arise in economics. A key difference is that economic models have forward looking optimizing agents and market clearing conditions. Our approach to handling borrowing constraints draws on Lu et al. (2021b) and Brzoza-Brzezina et al. (2015). We also build on the literature using neural networks to solve mean field games. Al-Aradi et al. (2022); Carmona and Laurière (2021) adapted the DGM to solve the PDE system arising in mean field games. Fouque and Zhang (2020); Carmona and Laurière (2022a); Germain et al. (2022b) proposed deep learning methods for mean field games and mean field control problems based either on direct approximation of the control or on an adaptation of the Deep BSDE method proposed by Han et al. (2018). We refer to e.g. Laurière (2021); Carmona and Laurière (2022b); Hu and Lauriere (2022) and the references therein for more details. However, these works are mostly focused on solving the problem at equilibrium and without aggregate shocks, so the equilibrium optimal control is learnt only for one distribution – the equilibrium distribution (or flow of distributions for non-stationary problems). Min and Hu (2021) proposed a deep learning method based on recurrent neural networks and signatures to solve mean field games with aggregate shocks, when the interactions are through moments and without solving the master equation. Perrin et al. (2022) introduced a deep reinforcement learning algorithm based on fictitious play to learn population-dependent policies for finite-state, finite-action mean field games. Two deep learning methods for finite state Master equations have been proposed and analyzed in Cohen et al. (2024), based on backward induction and the DGM respectively. In the context of mean field control, Carmona et al. (2019); Gu et al. (2021); Germain et al. (2022a); Frikha et al. (2023) used deep learning to compute social optima with controls depending on the population distribution, but these methods do not solve Nash equilibria. Relative the mathematics literature, our focus is on solving master equations for a class of mean field games with aggregate shocks.

This document is organised as follows. Section 2 describes the general economic environment that we will be studying and derives the master equation. Section 3 describes the different finite dimensional approximations to the master equation. Section 4 describes the solution approach. Section 5 applies our algorithm to continuous time version of Krusell and Smith (1998). Section 6 dicusses practical lessons. Section 7 concludes.

2 Economic Model

In this section, we outline the class of economic models for which our techniques are appropriate. At the high level, in economics terminology, we are solving continuous time, general equilibrium models with a distribution of optimizing agents who face idiosyncratic and aggregate shocks.[1] In mathematics terminology, we are solving mean field games with common noise.

[1]For ease of exposition, we restrict attention in the main text to models with one-dimensional aggregate shocks. The method is no different for the case with multi-dimensional aggregate shocks.

2.1 Environment

Setting: The model is in continuous time with infinite horizon. There is an exogenous one-dimensional aggregate state variable, z_t, which evolves according to:

$$dz_t = \mu^z(z_t)dt + \sigma^z(z_t)dB_t^0, \quad z_0 \text{ given}, \tag{2.1}$$

where B_t^0 denotes a common Brownian motion process. We let $\mathcal{F}_t^0$ denote the filtration generated by B_t^0.

Agent Problem: The economy is populated by a continuum of agents, indexed by $i \in I = [0,1]$. Each agent i has an idiosyncratic state vector, $x^i \in \mathcal{X}$, that evolves according to:

$$dx_t^i = \mu^x(c_t^i, x_t^i, z_t, q_t)dt + \sigma^x(x_t^i, z_t, q_t)dB_t^i + \gamma^x(x_t^i, z_t, q_t)dJ_t^i, \quad x_0^i \text{ given}, \tag{2.2}$$

where c_t^i is a one-dimensional control variable chosen by the agent, $q_t \in \mathcal{Q}$ is a collection of aggregate prices in the economy that will be determined endogenously in equilibrium, B_t^i denotes an N-dimensional idiosyncratic Brownian motion process, and J_t^i denotes an idiosyncratic Poisson jump process.[2] We let $\lambda(x)$ denote the rate at which Poisson jump shocks arrive given idiosyncratic state x.

Each agent i has a belief about the stochastic price process $\tilde{q} = \{\tilde{q}_t : t \geq 0\}$ adapted to $\mathcal{F}_t^0$. Given their belief, agent i chooses their control process, $c^i = \{c_t^i : t \geq 0\}$, to solve:

$$\begin{aligned} V(x_0^i, z_0) = \max_{c^i \in \mathcal{C}} \mathbb{E}_0 &\left[\int_0^\infty e^{-\rho t} u(c_t^i) dt \right] \\ s.t. \quad &(2.1),\ (2.2), \end{aligned} \tag{2.3}$$

where $\rho > 0$ is a discount parameter, $u(c_t^i)$ is the flow utility the agent gets and $\mathcal{C} = \{c_t^i \in \mathcal{C}(x_t^i, z_t, q_t) : t \geq 0\}$ is the set of admissible controls, where $\mathcal{C}(x, z, q)$ denotes the set of possible actions for a player whose current state is x, when the aggregate state is z and the prices are q. This constraint set incorporates any "financial frictions" that restrict agent choices. A classic example in economics is that the control must keep x_t^i positive. We let $\mathcal{X}$ denote the domain of x_t^i implicitly defined by the constraint that $c^i \in \mathcal{C}$. We assume that u is increasing and concave.

Distributions and Markets: We let $G_t = \mathcal{L}(x_t^i | \mathcal{F}_t^0)$ and g_t denote the population distribution and density across x_t^i at time t, for a given history $\mathcal{F}_t^0$. We assume that the economy contains a collection of markets with finite dimensional price vector q_t and market clearing

[2]We do not consider the case where dB_t^0 directly impacts the evolution of idiosyncratic states. In principle, the techniques outlined in this paper still apply in this case but such cases typically involve long term asset pricing that leads to more complicated market clearing conditions. We take up these issues in Gopalakrishna et al. (2024).

conditions that allow us to solve for q_t explicitly in terms of z_t and g_t[3]:

$$q_t = Q(z_t, g_t), \quad \forall t \geq 0, \tag{2.4}$$

We assume that markets are incomplete in the sense that agents cannot trade claims directly (or indirectly) on their idiosyncratic shocks dB_t^i and dJ_t^i. This means that the idiosyncratic shocks generate a non-degenerate cross sectional distribution of agent states.

Equilibrium: Given an initial density g_0, an equilibrium for this economy consists of a collection of $\mathcal{F}_t^0$-adapted stochastic processes, $\{c_t^i, g_t, q_t, z_t : t \geq 0, i \in I\}$, that satisfy the following conditions: (i) each agent's control process c^i solves problem (2.3) given their belief that the price process is $\tilde{q}$, (ii) the equilibrium prices q satisfy market clearing condition (2.4), and (iii) agent beliefs about the price process are consistent with the optimal behaviour of other agents in the sense that $\tilde{q} = q$.

2.2 Recursive Representation of Equilibrium

We assume that there exists an equilibrium that is recursive in the aggregate state variables: $\{z, g\}$. Observe that the price vector q can be expressed explicitly in terms of $\{z, g\}$ so beliefs about the price process can be characterized by beliefs about the evolution of the distribution $dg_t(x) = \tilde{\mu}^g(z_t, g_t)dt$.

Hamilton Jacobi Bellman Equation (HJBE): In principle, the constraint $c_t^i \in \mathcal{C}(x_t^i, z_t, q_t)$ could generate a boundary condition:

$$\Phi(x, z, g, V(x, z, g), D_x V(x, z, g)) \geq 0, \qquad x \in \partial\mathcal{X}.$$

However, in order to help the neural network train the value function, we replace the hard constraint on the set of admissible controls by a flow utility penalty $\psi(c, x, z, g)$ that is larger the "more" that $c_t^i \in \mathcal{C}(x_t^i, z_t, q_t)$ is violated. We provide explicit examples of ψ in our applications in Section 5. Given a belief about the evolution of the distribution, $\tilde{\mu}^g(z_t, g_t)$, the agent's optimal choice of control c solves the HJBE:

$$\begin{aligned} 0 = \max_{c \in \mathcal{C}(x,z,g)} \Big\{ & -\rho V(x, z, g) + u(c) + \psi(c, x, z, g) + D_x V(x, z, g)\mu^x(c, x, z, Q(z, g)) \\ & + \frac{1}{2}\mathrm{tr}\left\{\Sigma^x(x, z, Q(z, g)) D_x^2 V(x, z, g)\right\} + \lambda(x) \left(V(x + \gamma^x(x, z, q), z, g) - V(x, z, g)\right) \\ & + \partial_z V(x, z, g)\mu^z(z) + \frac{1}{2}\left(\sigma^z(z)\right)^2 \partial_{zz} V(x, z, g) \\ & + \int_{\mathcal{X}} \tilde{\mu}^g(z, g) \frac{\partial V}{\partial g}(x, z, g)(y) dy \Big\} \end{aligned} \tag{2.5}$$

[3]We will focus on numerical examples where $q_t = Q(z_t, \bar{g}_t)$, where $\bar{g}_t$ is the mean of g but this is not a constraint on the algorithm.

where $V(x,z,g)$ is the value function of the household, $\partial V/\partial g$ is the Frechet derivative of V with respect to the distribution, and $\Sigma^x(\cdot) := \sigma^x(\cdot)\,(\sigma^x(\cdot))^\top$. The intuition behind this HJBE is that, although the agent can only influence x, the state is (x,z,g) and hence the value function should take these variables as inputs. From V, the optimal consumption, denoted by c^*, can be computed for every (x,z,g), which allows a representative player to react optimally to any population distribution. The optimal control, c^*, is characterised by the first order condition:

$$0 = u'(c^*(x,z,g)) + \partial_c\psi'(c^*(x,z,g),x,z,g) + D_xV(x,z,g)\partial_c\mu^x(c^*(x,z,g),x,z,Q(z,g))$$

Kolmogorov Forward Equation (KFE): Denote the recursive equilibrium optimal control of the individual agents by $c^*(x,z,g;\tilde{\mu}^g)$. Compared with the above notation, we add $\tilde{\mu}^g$ to stress the fact that the belief of the agent may differ from the true μ^g. Then, for a given z path, the evolution of the distribution under the optimal control can be characterized by the Kolmogorov Forward Equation (KFE):[4]

$$\begin{aligned} dg_t(x) &= \mu^g(c^*(x,z_t,g_t;\tilde{\mu}^g),x,z_t,g_t)dt, \quad \text{where} \qquad (2.6)\\ \mu^g(c_t^*,x,z_t,g_t) :=& -\partial_x[\mu^x(c_t^*(x,z_t,g_t;\tilde{\mu}^g),x,z_t,Q(z_t,g_t))g_t(x)] + \frac{1}{2}\partial_{xx}[(\sigma^x(z_t))^2g_t(x)]\\ &+ \lambda((1-\gamma'(x,z_t,q_t)g_t(x-\gamma(x,z_t,q_t)) - g_t(x)) \end{aligned}$$

Under this recursive characterization, the belief consistency condition becomes that $\mu^g = \tilde{\mu}^g$.

Master Equation: We follow the approach of Lions (2011) and characterize the equilibrium in one PDE, which is often referred to as the "master equation" of the "mean field game". This formulation is particularly convenient when the evolution of the economy is subject to aggregate shocks and the evolution of the aggregate state variables cannot be determined deterministically. Conceptually, the master equation is related to the HJBE (2.5) but it imposes the belief consistency by putting the equilibrium KFE into the HJBE. In equilibrium, the value function $V(x,z,g)$ is the solution to the following "master equation":

$$(\mathcal{L}V)(x,z,g) = 0 \qquad (2.7)$$

[4]Observe that there is no noise in the KFE because dB_t^0 does not directly impact the evolution of idiosyncratic states.

where the operator $(\mathcal{L}V)(x,z,g) := (\mathcal{L}^h_{(c^*,Q)}V + \mathcal{L}^g V)(x,z,g)$ is defined by:

$$\begin{aligned}
(\mathcal{L}^h_{(c^*,Q)}V)(x,z,g) :=& -\rho V(x,z,g) + u(c^*(x,z,g)) + \psi(c,x,z,g) \\
&+ D_x V(x,z,g)\mu^x(c^*(x,z,g),x,z,Q(z,g)) \\
&+ \frac{1}{2}\text{tr}\left\{\Sigma^x(x,z,Q(z,g))D_x^2 V(x,z,g)\right\} \\
&+ \lambda(x)\left(V(x+\gamma^x(x,z,q),z,g) - V(x,z,g)\right) \\
&+ \partial_z V(x,z,g)\mu^z(z) + \frac{1}{2}\left(\sigma^z(z)\right)^2 \partial_{zz}V(x,z,g) \\
(\mathcal{L}^g V)(x,z,g) :=& \int_{\mathcal{X}} \mu^g(c^*(x,z,g),z,g)\frac{\partial V}{\partial g}(x,z,g)(y)dy.
\end{aligned}$$

In this notation, the $\mathcal{L}^h_{(c^*,Q)}$ operator reflects the optimization problem of the household and the $\mathcal{L}^g$ operator reflects how the evolution of the distribution affects the household value.[5] Intuitively, $V(x,z,g)$ can be interpreted as the optimal value of a representative player who starts at state x, with aggregate shock equal to z, and who faces a population that starts at the distribution g and then plays according to the Nash equilibrium control c^*. The technical challenge of working with the master equation is that it contains an infinite dimensional variable, g, and a derivative with respect to this variable. This poses a collection of mathematical difficulties for the mean field game theory literature, which attempts to find conditions under which the infinite dimensional master equation is well defined and has a solution. We refer to Cardaliaguet et al. (2015); Bensoussan et al. (2015) for more details. By contrast, we are focused on numerical approximation, which means that we need to find a finite dimensional approximation to the distribution, convert the master equation into a high, but finite, dimensional PDE, and develop techniques for solving the PDE. The goal of our paper is use deep learning techniques to characterize such a finite dimensional numerical approximation.

2.3 Model Generality

Our model set up nests many canonical models in macroeconomics such as Krusell and Smith (1998). However, we have also made a number of strong assumptions, which we discuss below:

(i). We have assumed that the prices, q, can be expressed explicitly in closed form as functions of the aggregate state variables (z,g), see (2.4). In models with more complicated market clearing conditions we need to solve an auxiliary fixed point problem in order to solve for the recursive representation of the price. Gopalakrishna et al. (2024) extends the methodology to resolve the challenges of introducing assets with complicated pricing.

[5]We define the gradient $D_x V(\cdots)$ as a row vector, such that the product $D_x V(\cdots)\mu^x(\cdots)$ has to be interpreted as an inner product for multi-dimensional x.

(ii). We have assumed that the distribution only enters the master equation through through the pricing function. If the model had search and matching frictions, then the distribution would impact which agents are likely to be matched and so enter the master equation in a more complicated way. Payne et al. (2024) extends the methodology to resolve these challenges.

(iii). We have assumed that the aggregate Brownian motion B^0 does not directly shock the evolution of idiosyncratic states. Instead it changes prices and so indirectly changes agent controls. This means that the KFE (2.6) does not have an aggregate noise term dB^0_t. The solution approach can easily be generalized to handle KFEs with aggregate noise so long as it does not lead to significantly more complicated market clearing conditions (a common extension studied is Fernández-Villaverde et al. (2023)). As discussed in points (i) and (ii), the greater difficulty is introducing assets with prices that are only an implicit function of the distribution or non-competitive markets.

Ultimately, the goal of this paper is not to offer a toolbox to solve every macroeconomic model. Instead, we explain and compare how to use different types of distribution approximations to solve continuous time models with heterogeneous agents. This offers a foundational set of techniques that are extended in other papers.

3 Finite Dimensional Master Equation

In this section, we study finite dimensional approximations to the population distribution g. Let $\hat{\varphi} \in \hat{\Phi} \subseteq \mathbb{R}^N$ be a finite dimensional parameter vector, which could represent the collection of agents, the bins in a histogram, or the coefficients in a projection. Let $\hat{G}$ be a mapping from parameters to distributions:

$$\hat{G} : \hat{\Phi} \to \mathcal{G}, \quad \hat{\varphi} \mapsto \hat{G}(\hat{\varphi}) = \hat{g}$$

We look for an approximation to the value function of the form:

$$\hat{V} : (\mathcal{X}, \mathcal{Z}, \hat{\Phi}) \to \mathbb{R}, \quad (x, z, \hat{\varphi}) \mapsto \hat{V}(x, z, \hat{\varphi})$$

that satisfies an approximate master equation:

$$0 = \hat{\mathcal{L}}\hat{V} = \hat{\mathcal{L}}^h\hat{V} + \hat{\mathcal{L}}^g\hat{V}$$

where $\hat{\mathcal{L}}^h$ is the approximate operator for the household optimization problem and $\hat{\mathcal{L}}^g$ is the approximate operator for how changes to the distribution approximation impact household value. The first term, $\hat{\mathcal{L}}^h$, is essentially $\mathcal{L}^h$ with the distributional approximation substituted in:

$$
\begin{aligned}
(\hat{\mathcal{L}}^h \hat{V})(x,z,\hat{\varphi}) := (\mathcal{L}^h_{(\hat{c}^*,\hat{Q})}\hat{V})(x,z,\hat{\varphi}) \\
= & -\rho\hat{V}(x,z,\hat{\varphi}) + u(\hat{c}^*(x,z,\hat{\varphi})) + \psi(\hat{c}^*(x,z,\hat{\varphi}),x,z,\hat{\varphi}) \\
& + D_x\hat{V}(x,z,\hat{\varphi})\mu^x(\hat{c}^*(x,z,\hat{\varphi}),x,z,\hat{Q}(z,\hat{\varphi})) \\
& + \frac{1}{2}\mathrm{tr}\left\{\Sigma^x(x,z,\hat{Q}(z,\hat{\varphi}))D_x^2\hat{V}(x,z,\hat{\varphi})\right\} \\
& + \lambda(x)\left(\hat{V}(x+\gamma^x(x,z,\hat{\varphi}),z,\hat{\varphi}) - \hat{V}(x,z,\hat{\varphi})\right) \\
& + \partial_z\hat{V}(x,z,\hat{\varphi})\mu^z(z) + \frac{1}{2}\left(\sigma^z(z)\right)^2\partial_{zz}\hat{V}(x,z,\hat{\varphi}) \qquad (3.1)
\end{aligned}
$$

where $\hat{Q}(z,\hat{\varphi}) := Q(z,\hat{G}(\hat{\varphi}))$ and $\hat{c}^*$ is defined to satisfy:

$$
\begin{aligned}
0 = & \, u'(\hat{c}^*(x,z,\hat{\varphi})) + \partial_c\psi(\hat{c}^*(x,z,\hat{\varphi}),x,z,\hat{\varphi}) \\
& + D_x\hat{V}(x,z,\hat{\varphi})\partial_c\mu^x(\hat{c}^*(x,z,\hat{\varphi}),x,z,\hat{Q}(z,\hat{\varphi})) \qquad (3.2)
\end{aligned}
$$

The second term, $\hat{\mathcal{L}}^g$, involves a more complicated change to the operator that depends on the way that the distribution is approximated.

We characterize three distribution approximation approaches. The first approach approximates the distribution with a finite collection of agents. The second approach approximates the continuous state space by a finite collection of grid points. The third approach projects the distribution onto a finite set of basis functions. In each case, we show how $\hat{G}$ and $\hat{\mathcal{L}}^g$ are defined.

3.1 Finite Agent Approximation

Distribution approximation: In this approach, we restrict the model so that the economy contains a large but finite number of agents $N < \infty$ agents. In this case, the finite dimensional parameter vector is the vector of agent positions:

$$\hat{\varphi}_t := \left(x_t^i\right)_{i\leq N}.$$

The mapping $\hat{G}$ takes the agents positions and computes the empirical measure:

$$\hat{G}(\hat{\varphi}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{x_t^i}$$

where $\delta_{x_t^i}$ denotes a Dirac mass at x_t^i. The evolution $\hat{\varphi}_t$ is simply the law of motion for each agents x_t^i, as described by equation (2.2).

The Operator $\hat{\mathcal{L}}^g$: The market clearing condition now becomes $q_t = Q(z_t, \hat{\varphi}_t) = Q(z, \hat{G}(\hat{\varphi}_t))$, as described in the introduction. However, to maintain the price taking assumption in the finite agent model, we impose that agent i behaves as if their individual actions do not influence prices. Formally, this means that agent i perceives the pricing function to be:

$$q_t = Q(z_t, \hat{\varphi}_t^{-i})$$

where $\hat{\varphi}_t^{-i} = \{x_t^j \in N^{-i}\}$ is the position of the other agents $N^{-i} := \{j \leq I : j \neq i\}$. Ultimately, this will ensure that the neural network trains the policy rules as if the agents believe that their assets do not influence the market prices.

Let $c^*(x^i, z, \hat{\varphi})$ denote the equilibrium optimal control. Let $\hat{V}(x^i, z, \hat{\varphi})$ denote the value function for the master equation in the economy with I price taking agents. Then $\hat{V}(x^i, z, \hat{\varphi})$ solves $(\hat{\mathcal{L}}\hat{V})(x^i, z, \hat{\varphi}) = 0$, where $\hat{\mathcal{L}}\hat{V} = \hat{\mathcal{L}}^h\hat{V} + \hat{\mathcal{L}}^g\hat{V}$, the (approximate) optimization operator $\hat{\mathcal{L}}^h$ is given by (3.1), and the (approximate) distribution impact operator $\hat{\mathcal{L}}^g$ is given by:

$$\begin{aligned}(\hat{\mathcal{L}}^g\hat{V})(x^i, z, \hat{\varphi}) := &\sum_{j\neq i} \frac{\partial \hat{V}}{\partial \hat{\varphi}^j}(x^i, z, \hat{\varphi})\mu^x(c^*(x^j, z, \hat{\varphi}), x^j, z, Q(z, \hat{\varphi}^{-j})) \\ &+ \sum_{j\neq i} \frac{1}{2}\mathrm{tr}\left\{\Sigma^x(x^j, z, Q(z, \hat{\varphi}^{-j}))D^2_{g^j}\hat{V}(x^i, z, \hat{\varphi})\right\} \\ &+ \sum_{j\neq i} \lambda(x^j)\left(\hat{V}(x^i, z, \{x^j + \gamma^x(x^j, z, Q(z, \hat{\varphi}^{-j})), \hat{\varphi}^{-j}\}) - \hat{V}(x^i, z, \hat{\varphi}^{-i})\right),\end{aligned} \tag{3.3}$$

The operator for the impact of distributional changes on the household, $\hat{\mathcal{L}}^g$, becomes finite dimensional because the economy only needs to track the evolution of a finite number of agents. The notation $\frac{\partial V}{\partial \hat{\varphi}^j}(x^i, z, \hat{\varphi})$ means that we take the partial derivative of V with respect to the j-th point in $\hat{\varphi}$, i.e., x_t^j.

Convergence properties: The solution $V(x^i, z, \hat{\varphi})$ is expected to converge to the solution of the master equation (2.7) as the number of agents, N, grows to infinity so long V is smooth. Intuitively, this relies on the idiosyncratic noise in the population distribution averaging out as the population becomes large, see e.g. Sznitman (1991). Such results have been extended to systems with equilibrium conditions by the mean field games literature; see e.g. Cardaliaguet et al. (2015); Lacker (2020); Delarue et al. (2020).

3.2 Discrete State Space Approximation

Distribution approximation: We consider N points in the state space, denoted by $x_1, \ldots, x_N \in \mathcal{X}$. We approximate g by a vector $\hat{\varphi} \in \mathbb{R}^N$, whose values represent the masses at $x_1, \ldots, x_N$.

The mapping $\hat{G}$ then takes the form:

$$\hat{G}(\hat{\varphi}) = \sum_{n=1}^{N} \hat{\varphi}_{n,t} \delta_{x_n}$$

where again δ_{x_n} denotes a Dirac mass at x^i. Conceptually, the finite agent approximation fixes the mass associated to each x_t^i and allows the x_t^i values to move whereas the discrete state space approximation fixes the grid points x^i and allows the masses at each grid point to move.

We determine $\hat{\mu}_{\hat{\varphi}}$ as an approximation to the KFE (2.6). The KFE under optimal control (2.6) is replaced by an ordinary differential equation in dimension N of the form:

$$d\hat{\varphi}_t = \hat{\mu}_{\hat{\varphi}}(z_t, \hat{\varphi}_t)dt \tag{3.4}$$

describing the evolution of mass at the values at $x_1, \ldots, x_N$. The right-hand-side needs to be obtained using information from the KFE (2.6). In our numerical examples we use a finite difference approximation to the KFE to derive $\hat{\mu}_{\hat{\varphi}}$, analogous to the approximation described in Achdou et al. (2022). However, the technique can be applied to other types of approximations, like the finite volume method used by Huang (2022). To be specific, let us consider the approximation of the KFE equation (2.6) using a finite difference scheme. The evolution of $\hat{\varphi}$, based on the evolution of g (assuming that the agent has the correct belief, i.e., $\tilde{\mu}^g = \mu^g$), is of the form (3.4) where, for $z \in \mathbb{R}$ and $\hat{\varphi} \in \mathbb{R}^N$, for every $n = 1, \ldots, N$,

$$\begin{aligned}\hat{\mu}_{\hat{\varphi},n}(z, \hat{\varphi}) := & -D_x[M_t(z, \hat{\varphi})\hat{\varphi}]_n + \frac{1}{2} D_x^2[(\sigma^x(z))^2 \hat{\varphi}]_n \\ & + \lambda((1 - \gamma'(x_n, z, \hat{Q}(z, \hat{\varphi}))\hat{\varphi}(x_n - \gamma(x_n, z, \hat{Q}(z, \hat{\varphi}))) - \hat{\varphi}_n),\end{aligned}$$

with $\hat{Q}(z, \hat{\varphi}) = Q(z, \hat{G}(\hat{\varphi}))$ and $M_t(z, \hat{\varphi}) \in \mathbb{R}^N$ is defined by:

$$M_{n,t}(z, \hat{\varphi}) = \mu^x(c_t^*(x_n, z, \hat{\varphi}; \hat{\mu}_{\hat{\varphi}}), x_n, z, \hat{Q}(z, \hat{\varphi}))$$

and the finite difference operators $D_x : \mathbb{R}^N \to \mathbb{R}^N$ and $D_x^2 : \mathbb{R}^N \to \mathbb{R}^N$ approximate respectively the first and the second order differential operators in space when the functions are represented by values at the points of the grid $x_1, \ldots, x_N$.

The operator $\hat{\mathcal{L}}^g$: The operator $\hat{\mathcal{L}}^g$ in the approximate master equation is defined by:

$$(\hat{\mathcal{L}}^g \hat{V})(x, z, \hat{\varphi}) = \sum_{n=1}^{N} \hat{\mu}_{\hat{\varphi},n}(z, \hat{\varphi}) \frac{\partial \hat{V}}{\partial \hat{\varphi}_n}(x, z, \hat{\varphi}), \tag{3.5}$$

where $\hat{\mu}_{\hat{\varphi}}(z, \hat{\varphi}) = (\hat{\mu}_{\hat{\varphi},1}(z, \hat{\varphi}), \ldots, \hat{\mu}_{\hat{\varphi},N}(z, \hat{\varphi}))$ is the vector characterizing the evolution of the approximate distribution on the state space, as described above.

Convergence properties: Once again, we expect the solution of the approximate master equation $V(x, z, \hat{g})$ to converge towards the solution of the true master equation (2.7) so long V is smooth. Convergence of mean field games with discrete state spaces to mean field games with continuous state spaces has been proved in Bayraktar et al. (2018); Hadikhanloo and Silva (2019) without noise or with idiosyncratic noise, and in in Bertucci and Cecchin (2022) with common noise.

3.3 Projection onto Basis

Distribution approximation: In this approach, we represent the distribution g_t by functions of the form

$$\hat{g}_t(x) = \hat{G}(\hat{\varphi}_t)(x) := b_0(x) + \sum_{n=1}^{N} \hat{\varphi}_{n,t} b_n(x),$$

where $\hat{\varphi}_t := (\hat{\varphi}_{1,t}, ..., \hat{\varphi}_{N,t}) \in \hat{\Phi} := \mathbb{R}^N$ is a vector of real-valued coefficients and $b_0, b_1, ..., b_N$ is a collection of linearly independent real-valued functions on $\mathcal{X}$ that satisfy

$$\int_{\mathcal{X}} b_n(x)dx = \begin{cases} 1, & n = 0 \\ 0, & n \geq 1 \end{cases} \tag{3.6}$$

and we refer to as the basis of the projection. To complete this approximation description we need to specify the the law of motion for $\mu_{\hat{\varphi}}$ and the choice of basis.

We derive $\mu_{\hat{\varphi}}$ as an approximation to the KFE (2.6). For the projection approach, we can flexibly focus the approximation accuracy on pre-selected statistics of the distribution. This is useful in economic problems where we know certain statistic of the distribution are particularly important for calculating prices through $q_t = Q(z_t, g_t)$. To make this precise, we start with the integral form of the KFE (2.6):

$$\begin{aligned} \frac{d \int_{\mathcal{X}} \phi(x) g_t(x) dx}{dt} &= \int_{\mathcal{X}} \phi'(x) \mu^x(c_t^*(x, z_t, g_t), x, z_t, Q(z_t, g_t)) g_t(x) dx \\ &+ \frac{1}{2} \int_{\mathcal{X}} \phi''(x) (\sigma^x(z_t))^2 g_t(x) dx \\ &+ \int_{\mathcal{X}} \lambda(x) \left(\phi\left(x + \gamma(x, z_t, Q(z_t, g_t))\right) - \phi(x) \right) g_t(x) dx \\ &=: \mu_\phi(z_t, g_t; c^*) \end{aligned}$$

that has to hold for all "test functions" $\phi : \mathcal{X} \to \mathbb{R}$ on a suitable space. This variant of the KFE describes the time evolution of all statistics $\int_{\mathcal{X}} \phi(x) g_t(x) dx$ of the distribution. For our approximation, we select M test functions $(\phi_1, ..., \phi_M)$ that describe statistics of interest where $M \geq N$. Given this selection, we define the coefficient drifts $\mu_{\hat{\varphi},n}(z, \hat{\varphi})$ (conditional on the aggregate state $(z, \hat{\varphi})$) as the regression coefficients that minimize, in the least-squares sense, the M linear regression residuals[6]

$$\varepsilon_m(z,\hat{\varphi}) := \mu_{\phi_m}(z,\hat{G}(\hat{\varphi});\hat{c}^*) - \sum_{n=1}^{N} \mu_{\hat{\varphi},n}(z,\hat{\varphi}) \int_X \phi_m(x) b_n(x) dx, \qquad m = 1, ..., M.$$

where $\mu_{\phi_m}(z,\hat{G}(\hat{\varphi});\hat{c}^*)$

The approximation presented so far works, in principle, for any choice of basis. Here we propose a basis that approximately tracks the persistent dimensions of g_t while neglecting those dimensions that mean-revert fast. These persistent dimensions of the distribution are related to certain eigenfunctions of the differential operator characterizing the KFE (2.6). Because this differential operator is generally time-dependent and stochastic, we first replace it by a time-invariant steady-state operator $\mathcal{L}^{KF,ss}$ defined as the KFE operator in a simplified model with common noise set to zero and under the assumption that the aggregate states have reached a steady state, $z_t = \bar{z}$ and $g_t = g^{ss}$.[7] Let $\{b_i : i \geq 0\}$ be the set of eigenfunctions of $\mathcal{L}^{KF,ss}$ with corresponding eigenvalues $\{\lambda_i \in \mathbb{R} : i \geq 0\}$. If the dynamics prescribed by the KFE are locally stable around the steady state g^{ss}, there is one eigenvalue $\lambda_0 = 0$ with eigenfunction $b_0 = g^{ss}$ and all remaining eigenvalues have negative real part, $\Re\lambda_i < 0$. We pick as our basis $b_0, b_1, ..., b_N$ the $N+1$ eigenfunctions corresponding to eigenvalues with real parts closest to zero as these represent the most persistent components. We provide further details on this basis choice in Appendix C.

The operator $\hat{\mathcal{L}}^g$: The operator $\hat{\mathcal{L}}^g$ in the approximate master equation is defined by

$$(\hat{\mathcal{L}}^g V)(x,z,\hat{\varphi}) = \sum_{n=1}^{N} \mu_{\hat{\varphi},n}(z,\hat{\varphi}) \frac{\partial V}{\partial \hat{\varphi}_n}(x,z,\hat{\varphi}).$$

Convergence: There are fewer results about convergence for projections in the mean-field-game literature. However, in discrete time, Prohl (2017) has proven convergence results for various projections.

3.4 Comparison

Table 1 summarizes the key differences between the distribution approximations: a finite population, a discrete state space, and a projection onto a finite set of basis functions.
We discuss how these approximations compare with regard to typical computational difficulties:

1. Dimensionality (N): The approximation dimension needs to be large enough to capture sufficient shape in the distribution. The projection method can potentially have

[6]In practice, the integral terms appearing in the residual formula have to be computed either analytically (if possible) or by numerical quadrature.

[7]This basis approximately tracks the persistent dimensions of g_t if $\mathcal{L}^{KF,ss}$ is similar to full stochastic operator $\mathcal{L}^{KF}$, which is plausible because they share many similar features.

	Finite Population	Discrete State	Projection
Distribution approx.	$\frac{1}{N}\sum_{i=1}^{N}\delta_{x_t^i}$	$\sum_{n=1}^{N}\hat{\varphi}_{n,t}\delta_{x_n}$	$\sum_{n=0}^{N}\hat{\varphi}_{n,t}b_n(x)$
KFE approx.	Evolution of other agents' states	Evolution of mass between discrete states	Evolution of projection coefficients

Table 1: Comparison of Distribution Approximations

the lowest dimension if the choice of basis is efficient ($N = 5$ in our examples). The finite population needs to be large enough to average out idiosyncratic noise ($N = 40$ in our examples). The discrete state space needs to be sufficiently fine to approximate the derivatives in the KFE, which means it needs a high dimension ($N = 200$ in our examples).

2. Customization: For the finite agent approach, we just choose N. For the discrete state space approach, we choose a grid and a method for approximating the KFE on the grid points. For the projection method, we choose a set of basis function and a set of statistics on which to minimize the error. In this sense, the projection is potentially lower dimensional because we need to make more intelligent choices in the setup.

3. Computational "bottlenecks": Each method has its own computational bottlenecks. For the finite agent method, we need to switch agent positions in the neural network approximation to V when we calculate the derivatives of V with respect to the other agent positions in equation (3.3). For the discrete state space method, the dimensionality of the approximation is the main computational problem. For the projection method, determining the drift $\mu_{\hat{\varphi}}$ of the distribution approximation is computationally involved as we need to solve a linear regression problem and compute several integrals by quadrature for every single evaluation of the master equation.

4. Nature of the distribution interaction: If agent decisions only depend upon a low dimensional moment of the distribution, then we find that using the finite agent approach (potentially coupled with dimension reduction) works well. This will be case in the variations on the Krusell and Smith (1998) that we study in this paper (and most other macro deep learning papers have studied). However, if the agent decisions really depend on the shape of the distribution, then other approximations look more attractive. For example, Payne et al. (2024) deploys these techniques in a Search and Matching framework using the discrete state space approximation.

4 Solution Approach

All approaches in section 3 lead to finite approximations to the density, $\hat{g}$, and the master equation operator $\hat{\mathcal{L}}$. However, the resulting master equations are high dimensional and so cannot be solved by traditional numerical techniques. Instead, we replace the solution to the approximate master equation by a neural network and deploy tools from the "deep learning" literature to "train" the neural network to solve the approximate master equation.

4.1 Neural Network Approximations

A neural network is a type of parametric functional approximation that is built by composing affine and non-linear functions in a chain or "network" structure (see Goodfellow et al. (2016) for a detailed discussion). We let $\hat{X} := \{x, z, \hat{\varphi}\}$ denote the collection of inputs into the approximate value function. We denote the neural network function to the value function by $V(\hat{X}) \approx \hat{V}(\hat{X}; \theta)$, where θ are the parameters in the neural network approximation that depend upon the architecture, i.e., the form of the approximation. There are many types of neural network approximations. The simplest form is a "feedforward" neural network which is defined by:

$$\begin{aligned}
h^{(1)} &= \phi^{(1)}(W^{(1)}\hat{X} + b^{(1)}) && \ldots\text{Hidden layer 1} \\
h^{(2)} &= \phi^{(2)}(W^{(2)}h^{(1)} + b^{(2)}) && \ldots\text{Hidden layer 2} \\
&\vdots \\
h^{(H)} &= \phi^{(H)}(W^{(H)}h^{(H-1)} + b^{(H)}) && \ldots\text{Hidden layer } H \\
o &= W^{(H+1)}h^{(H)} + b^{(H+1)} && \ldots\text{Output layer} \\
\hat{V} &= \phi^{(H+1)}(o) && \ldots\text{Output}
\end{aligned} \tag{4.1}$$

where the $\{h^{(i)}\}_{i \le H}$ are vectors referred to as "hidden layers" in the neural network, $\{W^{(i)}\}_{i \le (H+1)}$ are matrices referred to as the "weights" in each layer, $\{b^{(i)}\}_{i \le (H+1)}$ are vectors referred to as the "biases" in each layer, $\{\phi^{(i)}\}_{i \le (H+1)}$ are non-linear functions applied element-wise to each affine transformation and referred to as "activation functions" for each layer. The length of hidden layer, $h^{(i)}$, is defined as the number of neurons in the layer, which we refer to as $\#h^{(i)}$. The total collection of parameters is denoted by $\theta = \{W^{(i)}, b^{(i)}\}_{i \le (H+1)}$. The goal of deep learning is to train the parameters, θ, to make $\hat{V}(\cdot; \theta)$ a close approximation to V.

The neural network defined in (4.1) is called a "feedforward" network because hidden layer i cannot depend on hidden layers $j > i$. This is in contrast to a "recurrent" neural network where any hidden layer can be a function of any other hidden layer. It is called "fully connected" if all the entries in the weight matrices can be non-zero so each layer can use all the entries in the previous layer. In this paper, we will consider a fully connected "feedforward" network to be the default network. This is because these networks are the quickest to train and so we typically start by trying out this approach. However, there are applications where we find that more complicated neural network architectures are useful. In particular, we find that the type of recurrent neural network suggested by the Deep Galerkin Method in Sirignano and Spiliopoulos (2018) is helpful for discrete state and projection approximations.

4.2 Solution Algorithm

We train the neural network to learn parameters θ that minimize the error in the master equation and boundary conditions. We describe the key steps in Algorithm 1.[8] Essentially, the algorithm samples random points in the discretized state space $\{x, z, \hat{\varphi}\}$, then calculates the master equation on those points under the current neural network approximation, and then updates the neural network parameters to decrease the error in the master equation. In fact, the loss consists of two terms: $\mathcal{E}^e$ for the PDE residual and $\mathcal{E}^s$, which is used to incorporate information about the shape of the solution (e.g., monotonicity or concavity). Specific examples of these functions will be discussed in the following sections. In the deep learning literature, this approach is sometimes referred to as “unsupervised” learning (e.g. Azinovic et al. (2022)) because we do not have direct observations of the value function, $V(x, z, \hat{g})$, and instead have to learn the value function indirectly via the master equation.

Although the algorithm is straightforward to describe at the high level, implementing the deep learning training scheme successfully involves a lot of complicated decisions. We discuss some of the key details below.

4.2.1 Sampling

A very important implementation aspect of our solution algorithm is the approach used to sample the set of training points S in each iteration of the algorithm. Sampling ultimately has to be tailored to the specific application at hand, possibly by experimenting with various options. This is a fundamental difference between deep learning for continuous time and discrete time techniques. Discrete time models need to calculate expectations and so typically need to use simulation to approximate the expectation operator. Continuous time models replace the expectation term in the Bellman equation by the derivative terms in the HJBE and then sample points on which to evaluate the HJBE. This gives continuous time techniques more flexibility in how to sample but can also make the sampling task harder. The following general considerations are relevant when deciding on how to sample.

First of all, we can treat the three components of training points, the idiosyncratic state x, the aggregate exogenous state z, and the distribution state $\hat{\varphi}$ separately by sampling them independently. The separation between x and $\hat{\varphi}$ deserves particular emphasis in the

[8] The generic pseudo-code given in Algorithm 1 can be modified in practice. For example, instead of fixing a precision threshold, one can fix a number of iterations, and instead of using a fixed sequence of learning rates, one can use an adaptive method, such as Adam.

Algorithm 1: Psuedo Code for Generic Solution Algorithm

```
Input   : Initial neural network parameters $\theta^0$, number of sample points $M$,
          positive weights $\kappa^e$ and $\kappa^s$ on the master equation errors; sequence of
          learning rates $\{\alpha_n : n \geq 0\}$, precision threshold $\epsilon$
Output: A neural network approximation $(x, z, \hat{\varphi}) \mapsto \hat{V}(x, z, \hat{\varphi}; \theta)$ of the value
          function.
1: Initialize neural network object $\hat{V}(x, z, \hat{\varphi}; \theta)$ with parameters $\theta$.
2: while Loss > $\epsilon$ do
3:     Generate $M$ new sample points, $S^n = \{(x_m, z_m, \hat{\varphi}_m)\}_{m \leq M}$.
4:     Calculate the weighted average error:
           $$\mathcal{E}(\theta^n, S^n) = \kappa^e \mathcal{E}^e(\theta^n, S^n) + \kappa^s \mathcal{E}^s(\theta^n, S^n)$$
       where $\mathcal{E}^e$ is the master equation error and $\mathcal{E}^s$ is a penalty for a "wrong" shape. $\mathcal{E}^e$ is
       taken to be the mean-squared error:
           $$\mathcal{E}^e(\theta^n, S^n) := \frac{1}{|S^n|} \sum_{(x,z,\hat{\varphi}) \in S^n} |(\hat{\mathcal{L}}\hat{V}(x, z, \hat{\varphi}; \theta^n))|^2$$
       where the derivatives in the operator $\hat{\mathcal{L}}$ are calculated using automatic
       differentiation. The definition of $\mathcal{E}^s$ depends on the problem (see examples in the
       next sections).
5:     Update the parameters using "stochastic gradient descent":
           $$\theta^{n+1} = \theta^n - \alpha_n D_\theta \mathcal{E}(\theta^n, S^n)$$
       where $D_\theta \mathcal{E}$ is the gradient (i.e., vector differential) operator.
6: end while
```

context of the finite agent approximation, for which also $\hat{\varphi}$ contains a sample of x-points (for the other agents). This is because we can focus the sampling on regions of the idiosyncratic state space with high curvature without having to simulate a lot of distributions that have agents hitting constraints. This is an important advantage to simulation based approaches that learn regions of high curvature by increasing the sample size to get enough agents in the high curvature regions in the simulation time series.

Sampling x and z is less complicated because their dimension is usually relatively low. For example, in macro models a typical dimension of x is 2 and a typical dimension of z is 1. For these variables, we typically sample from a pre-specified statistical distribution such as a uniform or normal distribution. The sampling can be refined by adopting a strategy called "active sampling" (see, e.g., Gopalakrishna (2021) and Lu et al. (2021a)) that adapts the sampling during training to actively learn in regions where the algorithm is having trouble minimizing the loss function. This is achieved by regularly inspecting the losses during training and adding training points to regions with the largest losses.

Sampling the distribution approximation $\hat{\varphi}$ is significantly more complicated. This is because it is typically high-dimensional and so we can only train the neural network on a very small subset of the total possible distributions. In this sense, deep learning does not break the "curse of dimensionality". Instead, it gives flexibility to train on a useful subspace that gives enough information to the value function for economically relevant distributions. This means that choosing the right subspace to sample on is very important for the algorithm to converge in reasonable time (or at all). Ultimately, this requires us to use some information about the model solution in the sampling. We have focused on the following three sampling schemes that use different information from the model solution:

(i) *Moment sampling:* We first draw samples for selected moments of the distribution that are important for calculating prices $\hat{Q}(z, \hat{\varphi})$. We then sample $\hat{\varphi}$ from a distribution that satisfies the moments drawn in the first step. For example, in many macroeconomic model, the mean of the distribution is particularly important for calculating prices. In this case, we would sample from the mean and then draw $\hat{\varphi}$ from a distribution with that mean. This final step could involve sampling from pre-specified distribution (e.g. uniform) or from an ergodic distribution, as described in (iii).

(ii) *Mixed steady state sampling:* We first solve for the steady state for a collection of fixed aggregate states z. This needs to be done only once before training begins. We then draw in each training step random mixtures of this collection of steady state distributions. In an optional final step, we introduce additional random variation in the sampling of $\hat{\varphi}$ by adding perturbations drawn from a pre-specified distribution (e.g. uniform).[9]

(iii) *Ergodic sampling:* We adapt the training sample dynamically by regularly simulating

[9]Without these perturbations, the random mixtures remain strictly confined to a subspace whose dimension (the number of steady states in the collection) is typically much smaller than $\dim \hat{\Phi}$.

the model economy based on the candidate solution for the value function from a previous iteration.

Two additional issues arise in sampling schemes that adapt the training sample dynamically, such as active sampling (for x and z) and ergodic sampling (for $\hat{\varphi}$). First, these schemes only adapt the sampling distribution in a meaningful way if the current guess for the value function is sufficiently good. It is therefore advisable to start with a pre-specified sampling distribution in early training and switch to a dynamic sampling scheme later on. Second, dynamically adapting the sample might lead to instability of training due to feedback effects between the training sample and the trained solution. We have found this issue to be particularly relevant for ergodic sampling. To mitigate the issue, a combination of two remedies has worked well in our numerical experiments: (i) to use ergodic sampling only for a fraction of the training sample and (ii) to update training points by simulating over a small time interval frequently instead of simulating over a long time interval infrequently.[10]

At a high level, we have found the following sampling strategies are useful for the different types of distribution approximations. For the finite agent approximation, we found moment sampling to be simple and effective because the $\hat{\varphi}$ variables have a natural interpretation as the idiosyncratic (x) states of the other agents in the population and so it is straightforward to determine a region of "typical" values to sample from. For the discrete state space approximation, we found the most stable approach was to start with mixed steady state sampling and move to ergodic sampling once the neural network started to converge. For the projection approximation, we found that a combination of moment sampling and ergodic sampling was effective. To put moment sampling to work with projections requires a rotation of the basis functions so as to isolate components that correspond to the selected moments, see Appendix A.2 for details. We discuss all three sampling strategies in detail for our Krusell and Smith (1998) example in Appendix A.3.2.

4.2.2 Extensions

In this subsection we consider two extensions to our algorithm that are potentially relevant for economic problems.

Boundary conditions: Algorithm 1 is suitable for solving a problem that does not include boundary conditions. This is sufficient for our purposes because we have "softened" any hard constraints in the problem by replacing them with a utility flow penalty (see Section 2.2). In principle, the solution algorithm could be extended to a problem that does require boundary conditions. In this case, we would need to sample an second sample S^b of training points on the boundary in step 3 and, in step 4, add an additional error term $\mathcal{E}^b$ (with corresponding

[10]In particular, despite the name "ergodic sampling", we do not insist on drawing training points from the ergodic distribution implied by the current value function candidate. Instead, the ergodic distribution is reached only gradually once the value function is close to convergence.

weight κ^b) for the mean-squared error of boundary condition residuals evaluated at the training set S^b. All other aspects of the algorithm would remain unchanged.

That being said, while conceptually straightforward, we have found in the context of our example model that the inequality boundary conditions arising from hard constraints represent a significant difficulty for the robustness of our solution algorithm. Specifically, we observed that the neural network only learns an accurate solution if the weights on equation residuals (κ^e) and the boundary condition (κ^b) in the loss function are well-calibrated. Replacing the hard constraints by a soft penalty makes the method much more robust.

Separate network for policy function $\hat{c}^*$ and staggered updating: In Algorithm 1, when evaluating the master equation residuals to determine $\mathcal{E}^e$, we compute the policy $\hat{c}^*$ as a function of $\hat{V}(\cdot;\theta)$ according to equation (3.2). There is therefore no need to parameterize the policy $\hat{c}^*$ with a separate neural network. However, there may be reasons to include a separate neural network for $\hat{c}^*$, as suggested by Duarte (2018).

First, if equation (3.2) does not have a closed form solution, the computational cost of solving this equation numerically for every point in the training sample (and in every iteration) may be prohibitively high. In this case, we could include a separate neural network for $\hat{c}^*$ with separate parameters θ^c and use this network in place of the true solution to equation (3.2) to evaluate the master equation residuals. This would necessitate training of the $\hat{c}^*$-network as well by adding, after step 5, one or several iterations to train θ^c to solve the algebraic equation (3.2). Each of these iterations involves steps analogous to steps 3, 4, and 5 in Algorithm 1.

Second, even if equation (3.2) can be solved, adding a separate neural network for $\hat{c}^*$ allows us to slow down the updating of the policy function akin to a "Howard improvement algorithm". To do so, we can make the updating of θ^c infrequent, effectively fixing the same policy rule $\hat{c}^*$ for several iterations in the training of θ. In some implementations for our numerical example we use this variant of our baseline algorithm. We have found that this can help with stability, particularly when starting from a poor (e.g. random) initial guess.

4.2.3 Other Features

There are some features of the algorithm that are typical to deep learning problems but less common for other techniques in economics so we address them here:

(i). *Why do we draw a new sample each epoch rather than fixing a sample from the start?* We find that fixing the sample across epochs leads to overfitting problems where the neural network matches the master equation solution at the sample points well but interpolates poorly in the rest of the state space.

(ii). *Could the same algorithm be run with an alternative parametric approximation such as Chebyshev polynomials?* In principle, it is possible. But, the key features of the training algorithm are that we need to be able to calculate automatic derivatives and solve a high dimensional non-linear optimization problem. The machine learning literature has invested heavily in getting non-linear optimizers to work well with neural networks and automatic differentiation. We are borrowing from these developments.

(iii). *Does this algorithm solve for the global or local minimum?* The stochastic gradient descent algorithm (or one of its variants, such as Adam) calculates the loss on random collections of points and so has some ability to wander the parameter space looking for the global minimum. However, it is often sufficient to find a local minimum that is a reasonably approximation.

(iv). *Why do we need shape constraints?* We find that deep learning algorithms can "cheat" by finding "bad" approximate solutions. For example, they are likely to find solutions where the value function has zero derivative with respect to dimensions where there is limited curvature. More generally, it seems that in some instances there are local minima which are quite easily learnt by the neural network and yet are very different from the true solution. We find that enforcing shape constraint such as monotonicity or concavity helps prevent the algorithm getting stuck at these solutions.

(v). *What about introducing a false time transient?* Gopalakrishna (2021) proposes to introducin a "false" time-step into the problem for the purpose of eliminating "cheat" solutions. We have not found this necessary (or computationally implementable for high dimensional models). Instead, we find that using shape constraints to eliminate bad approximate solutions works well in our case.

(vi). *What about imposing symmetry and/or dimension reduction?* Han et al. (2021) and Kahou et al. (2021) suggest feeding the distribution through a preliminary neural network that reduces the dimension and imposes symmetry; see also Germain et al. (2022a) in the case of social optima. In our numerical experiments with finite agent approximation, we find that we can solve the problem with and without this approach. However, we do find that when we impose dimension reduction the accuracy of our solution is very sensitive to how much dimension reduction is done. For the discrete state space method, we find it is important to impose a form of dimension reduction in order to keep the size of the neural network manageable.

5 Example: Uninsurable Income Risk and TFP Shocks

A canonical macroeconomic model with heterogeneous agents and aggregate risk is Krusell and Smith (1998), which we refer to as the KS model. In this section, we illustrate how our three solution approaches can be deployed to solve the model.

5.1 Model Specification

In this subsection, we briefly explain how the KS model fits into our general framework. We have included a more detailed derivation of the master equations for the Aiyagari (1994) and Krusell and Smith (1998) models in our: "Online Appendix: Krusell and Smith (1998) Model".

Setting: There is a perishable consumption good and a durable capital stock. The economy consists of a unit continuum $I = [0, 1]$ of households and a representative firm. The representative firm controls the production technology, which produces consumption goods according to the production function:

$$Y_t = e^{z_t} K_t^{\alpha} L_t^{1-\alpha}$$

where K_t is the capital rented at time t, L_t is the labour hired at time t, and z_t is the aggregate productivity, which follows an Ornstein-Uhlenbeck process:

$$dz_t = \eta(\overline{z} - z_t)dt + \sigma dB_t^0$$

with lower and upper reflecting boundaries at z_{min} and z_{max} respectively.

Households: Each household $i \in [0, 1]$ has discount rate ρ and gets flow utility $u(c_t^i) = (c_t^i)^{1-\gamma}/(1-\gamma)$ from consuming c_t^i consumption goods at time t. Each household has two idiosyncratic states $x_t^i = [a_t^i, l_t^i]$, where a_t^i is the household's net wealth and $l_t^i \in \{l_1, l_2\}$ is the household's labor endowment, where $n_1 < l_2$ so l_1 is interpreted as unemployment and l_2 is interpreted as employment. Labor endowments switch idiosyncratically between l_1 and l_2 at Poisson rate $\lambda(l_t^i)$. Households choose consumption c_t^i and their idiosyncratic state evolves according to:

$$dx_t^i = d\begin{bmatrix} a_t^i \\ l_t^i \end{bmatrix} = \begin{bmatrix} s(a_t^i, l_t^i, c_t^i, r_t, w_t) \\ 0 \end{bmatrix} dt + \begin{bmatrix} 0 \\ \check{l}_t^i - l_t^i \end{bmatrix} dJ_t^i$$

where r_t is the return on household wealth, w_t is the wage rate, $\check{l}_t^i$ is the complement of l_t^i, J_t^i is an idiosyncratic Poisson process with arrival rate $\lambda(l_t^i)$, and the agent's saving function is given by:

$$s(a, l, c, r, w) = wl + ra - c.$$

So, g_t denotes the population density across $\{a_t^i, l_t^i\}$ at time t, given a filtration $\mathcal{F}_t^0$ generated by the sequence of aggregate productivity shocks.

Assets, markets, and financial frictions: Each period, there are competitive markets for goods, capital rental, and labor. We use goods as the numeraire. We let r_t denote the rental rate on capital, w_t denote the wage rate on labor, and $q_t = [r_t, w_t]$ denote the price vector. Given g_t and z_t, firm optimization and market clearing imply that, given g_t, the prices r_t and w_t solve:

$$r_t = e^{z_t}\partial_K F(K_t, L) - \delta, \qquad w_t = e^{z_t}\partial_L F(K_t, L),$$
$$K_t = \sum_{j\in\{1,2\}} \int_{\underline{a}}^{\infty} a g_t(a, l_j)da \qquad L = \sum_{j\in\{1,2\}} \int_{\underline{a}}^{\infty} l_j g_t(a, l_j)da.$$

So, in the terminology of Section 2, we write the prices explicitly as functions of (g_t, z_t):

$$q_t = \begin{bmatrix} r_t \\ w_t \end{bmatrix} = \begin{bmatrix} \partial_K F\left(\sum_{j\in\{1,2\}} \int_{\underline{a}}^{\infty} a g_t(a, l_j)da, L\right) - \delta \\ \partial_L F\left(\sum_{j\in\{1,2\}} \int_{\underline{a}}^{\infty} a g_t(a, l_j)da, L\right) \end{bmatrix} =: Q(g_t, z_t) \tag{5.1}$$

Asset markets are incomplete so households cannot insure their idiosyncratic labor shocks. Instead, households can trade claims to the aggregate capital stock in a competitive asset market. The original Krusell and Smith (1998) model imposes the "borrowing constraint" that each agent's net asset position, a_t^i, must satisfy $a_t^i \geq \underline{a}$, where $\underline{a}$ is an exogenous "borrowing limit". This generates an inequality boundary constraint and a mass point, as discussed in Achdou et al. (2022). However, this causes difficulties for the neural network. So, to make the problem more tractable, we instead follow Brzoza-Brzezina et al. (2015) and introduce a penalty function ψ at the left boundary, replacing the agent flow utility by:

$$U(a_t, c_t) = u(c_t) + \mathbf{1}_{a_t \leq \underline{a}}\psi(a_t).$$

The penalty function we use here is the quadratic function: $\psi(a) = -\frac{1}{2}\kappa(a - \underline{a})^2$ where κ is a positive constant.

Master equation: Let $c^*(a, l, g, z)$ denote the equilibrium optimal household control. Then, the master equation for the ABH model is given by the following:

$$0 = (\mathcal{L}V)(a, l, z, g) = (\mathcal{L}^h V)(a, l, z, g) + (\mathcal{L}^g V)(a, l, z, g),$$

where the operators $\mathcal{L}^h$ and $\mathcal{L}^g$ are defined by:

$$\begin{aligned}
(\mathcal{L}^h V)(a,l,z,g) :=\ & -\rho V(a,l,z,g) + u(c^*(a,l,z,g)) + \mathbf{1}_{a\leq \underline{a}}\psi(a) \\
& + \partial_a V(a,l,z,g) s(a,l,c^*(a,l,z,g), r(g), w(g)) \\
& + \lambda(y)(V(a,\check{l},z,g) - V(a,l,z,g)) \\
& + \partial_Z V(a,l,z,g)\eta(\overline{z}-z) + \frac{1}{2}\sigma^2 \partial_{zz} V(a,l,z,g) \\
(\mathcal{L}^g V)(a,l,z,g) :=\ & \int_{\overline{a}}^{\infty} \frac{\partial V}{\partial g}(a,l,z,g)(b)\left(\lambda(\tilde{l})g(b,\tilde{l}) - \lambda(y)g(b,l)\right) db \\
& + \int_{\overline{a}}^{\infty} \partial_b \frac{\partial V}{\partial g}(a,l,g)(b) s\left(a,l,c^*(a,l,z,g), r(g,z), w(g,z)\right) g(b,l) db,
\end{aligned}$$

where $\check{l}$ denotes the complement of l, where $r(g)$ and $w(g)$ solve the system of equations (5.1), and the optimal consumption policy satisfies the first order optimality condition:

$$\partial_a V(a,l,z,g) = u'(c^*(a,l,z,g)).$$

In the next sections, we solve this master equation numerically using Algorithm 1. Because the optimal control is a function of the $\partial_a V(a,l,z,g)$, it will turn out to be more convenient to solve the master equation for the partial derivative, which we denote by $W(a,l,z,g) := \partial_a V(a,l,z,g)$. The parameters that we use in numerical experiments are in Appendix A.1.

5.2 Implementation Details

The implementation details for all methods are summarized in Table 2. Here we discuss some key features about the neural network structure, the sampling, and the loss function. We provide a more detailed description of the implementation in Appendix A.3.

First, consider the neural network structure. For the finite population approximation, we use a "plain vanilla" fully connected feed-forward neural network with 5 layers and 64 neurons per layer. We choose a tanh activation function in all hidden layers and a softplus activation in the output layer to ensure that the output is always positive. For the remaining two approximation methods, we use an architecture that combines the one proposed by Sirignano and Spiliopoulos (2018) ("recurrent") and a fully connected feed-forward network ("embedding"). The latter network is used to preprocess the distribution state $\hat{\varphi}$ before the result is passed to the main (recurrent) network. The recurrent portion of the network has 3 layers and 100 neurons per layer. The embedding portion has an output dimension of 10, 2 layers, and 128 neurons per layer for the discrete state space method and 64 neurons per layer for the projection method. We apply an elu activation in the last layer of the recurrent network to ensure positivity of the output. In addition, we multiply the neural network output by the factor $(a_0 + a)^{-\tilde{\eta}}$, where $a_0, \tilde{\eta} \geq 0$ are non-trainable shape

	Finite Population	Discrete State	Projection
Neural Network			
(i) Structure	Fully connected feed-forward	Recurrent with embedding	Recurrent with embedding
(ii) Activation, $(\phi^{(i)})_{i\leq H}$	tanh	tanh	tanh
(ii) Output, $\phi^{(H+1)}$	soft-plus	elu activation and factor $(a_0+a)^{-\tilde{\eta}}$	elu activation and factor $(a_0+a)^{-\tilde{\eta}}$
(iii) Layers, H	5	recurrent: 3 embedding: 2	recurrent: 3 embedding: 2
(iv) Neurons, $\#\lvert h\rvert$	64	recurrent: 100 embedding: 128	recurrent: 100 embedding: 64
(v) Initialization	$W(a,\cdot)=e^{-a}$	random	random
(vi) Auxiliary networks	none	consumption	consumption
Sampling			
(i) (a,l)	Active sampling $[\underline{a},\overline{a}]\times\{y_1,y_2\}$	Uniform sampling $[\underline{a},\overline{a}]\times\{y_1,y_2\}$	Uniform sampling $[\underline{a},\overline{a}]\times\{y_1,y_2\}$
(ii) $(\hat{\varphi}_i)_{i\leq N}$	Moment sampling: sample r and then random distribution of agents to generate r	Mixed steady-state sampling ergodic sampling	Moment sampling: sample K and then orthogonal coefficients uniformly ergodic sampling
(iii) z	$U[z_{min},z_{max}]$	$U[z_{min},z_{max}]$	$U[z_{min},z_{max}]$
Loss Function			
(i) Master equation	$\mathcal{E}^e$	$\mathcal{E}^e$	$\mathcal{E}^e$
(ii) Constraints	$\partial_a W(a,\cdot)<0$ and $\partial_z W(a,\cdot)<0$	$\partial_a W(a,\cdot)<0$ and $\partial_z W(a,\cdot)<0$	$\partial_a W(a,\cdot)<0$ and $\partial_z W(a,\cdot)<0$
(iii) Weights	$\kappa^e=100$, $\kappa^s=1$	$\kappa^e=\kappa^s=1$	$\kappa^e=\kappa^s=1$
Training			
(i) Learning rate	10^{-4}	Decaying from 3×10^{-4} to 10^{-6}	Decaying from 3×10^{-4} to 10^{-6}
(ii) Optimizer	ADAM	ADAM	ADAM

Table 2: Key Implementation Details

parameters. The additional factor aids training by reducing the amount of curvature in the marginal value function that must be captured by the neural network. Finally, in both the discrete state and the projection method, we use the variant of our solution algorithm with staggered updating of the consumption function and parameterize the latter by a separate auxiliary neural network that has the same structure as the one for the marginal value function with the exception that we do not multiply the network output by $(a_0 + a)^{-\tilde{\eta}}$ in the output layer.

Second, consider the sampling. For the finite agent approach, we use moment sampling. As we discuss in Subsections 5.5 and 5.6, this allows us to consider unanticipated shocks and the introduction of parameters as auxiliary inputs for the purposes of calibration. For the other techniques, we need to partially rely on ergodic sampling, which makes it very difficult to consider similar kinds of experiments.

For the discrete state approach, our main approach is ergodic sampling. As discussed in Section 4.2.1, this sampling scheme is only meaningful once a sufficiently good guess of the value function exists. We therefore start with mixed steady state sampling in early training and gradually increase the fraction of the training set taken from the ergodic sample.

For the projection approach, we use a variant of moment sampling for sampling capital stocks K from a uniform distribution. To be able to do this, we rotate the basis, such that the first basis vector points in the direction of increasing first moments (in the a-dimension) and all other basis vectors are orthogonal to the first. This rotation leaves the space of approximate distributions unaffected but leads to a one-to-one relationship between the aggregate capital stocks and the first component of the distribution state $\hat{\varphi}$. For the remaining components of $\hat{\varphi}$ we use a combination of uniform and ergodic sampling and gradually increase the fraction of the training set taken from the ergodic sample.

Third, consider the loss function. In all cases, we impose concavity constraints on V, which are equivalent to monotonicity constraints on W. In addition, we found it was very helpful to impose constraints on $\partial_{az} V$. This is because KS model actually has limited curvature with respect to z and so the Neural network tended to find approximate solutions where the z component was ignored.

Finally, consider the neural network training. In all cases, we choose an ADAM optimizer to perform the gradient descent steps. We fix the learning rate at 10^{-4} throughout for the finite agent approach, whereas we use a learning rate schedule with decaying learning rate (from 3×10^{-4} to 10^{-6}) for the remaining approaches.

5.3 Mean Reverting Aggregate Productivity Process

We solve the Krusell-Smith model using the three methods. The error in the master equation is shown in Table 3 below. Evidently all approaches have master equation losses to the approximate order of 10^{-5}.

	Master equation training loss
Finite Agent NN	3.037×10^{-5}
Discrete State Space NN	9.639×10^{-5}
Projection NN	8.506×10^{-6}

Table 3: Neural Networks' results for solving master equations with aggregate shocks. The master equation loss is $\mathcal{E}^e$ evaluated on sampled data generated in Appendix A.3.2.

Because there are aggregate shocks in the Krusell-Smith model, we do not have a clear benchmark because there is no traditional technique that provides an accurate solution to the model with aggregate shocks. However, we can compare to widely used approximation techniques in the literature. In particular, we compare to the approach suggested by Fernández-Villaverde et al. (2023), which uses a neural network to approximate a statistical law of motion but solves the Master equation using a finite difference scheme.

We compare to Fernández-Villaverde et al. (2023) by computing sample paths from all of our solution approaches. For the discrete state space and projection methods, we can generate sample paths by iterating the approximate KFE. For the finite agent method, computing the sample path is more complicated. We describe how we generate sample paths for the neural network solution in Algorithm 2 below. Essentially, we draw a series of productivity shocks from the Ornstein-Uhlenbeck process: $dz_t = \eta(\bar{z} - z)dt + \sigma dB_t^0$ and then evolve the population distribution by adding z_t to Algorithm 3 for the ABH model described in Section 5.5.

Figure 1 offers a visual inspection of the difference between our neural network solution and Fernández-Villaverde et al. (2023) for a particular path of productivity shocks. The upper-left panel shows the draw from the Ornstein-Uhlenbeck process: $dz_t = \eta(\bar{z} - z)dt + \sigma dB_t^0$. The upper left compares the evolution of capital stock. The second row plots compare the evolution of prices. The third and fourth row plots compare the population distribution at various times in the distribution. As can be seen in Figure 1, we get a similar path for aggregate capital stock, interest rates, wage rates, and the population distribution across all the methods.

In Figure 2, we generate multiple random TFP paths, z_t, and show the evolution of our Neural Network solution and solution in Fernández-Villaverde et al. (2023) in a "fan chart" that displays percentiles for the evolution of the population. In particular, we generate 1000 TFP paths starting from $z_0 = 0$ and calculate the corresponding aggregate capital evolution paths. We compute capital at different time t, sort to get the pth-quantile and plot the time series of the quantiles. Evidently, the finite agent and projection techniques are a close match to the Fernández-Villaverde et al. (2023). The discrete state space approach does a good job at lower quantiles but has more difficulty at the extremes. In general, our experience is that the discrete state space approximation was most difficult to work with.

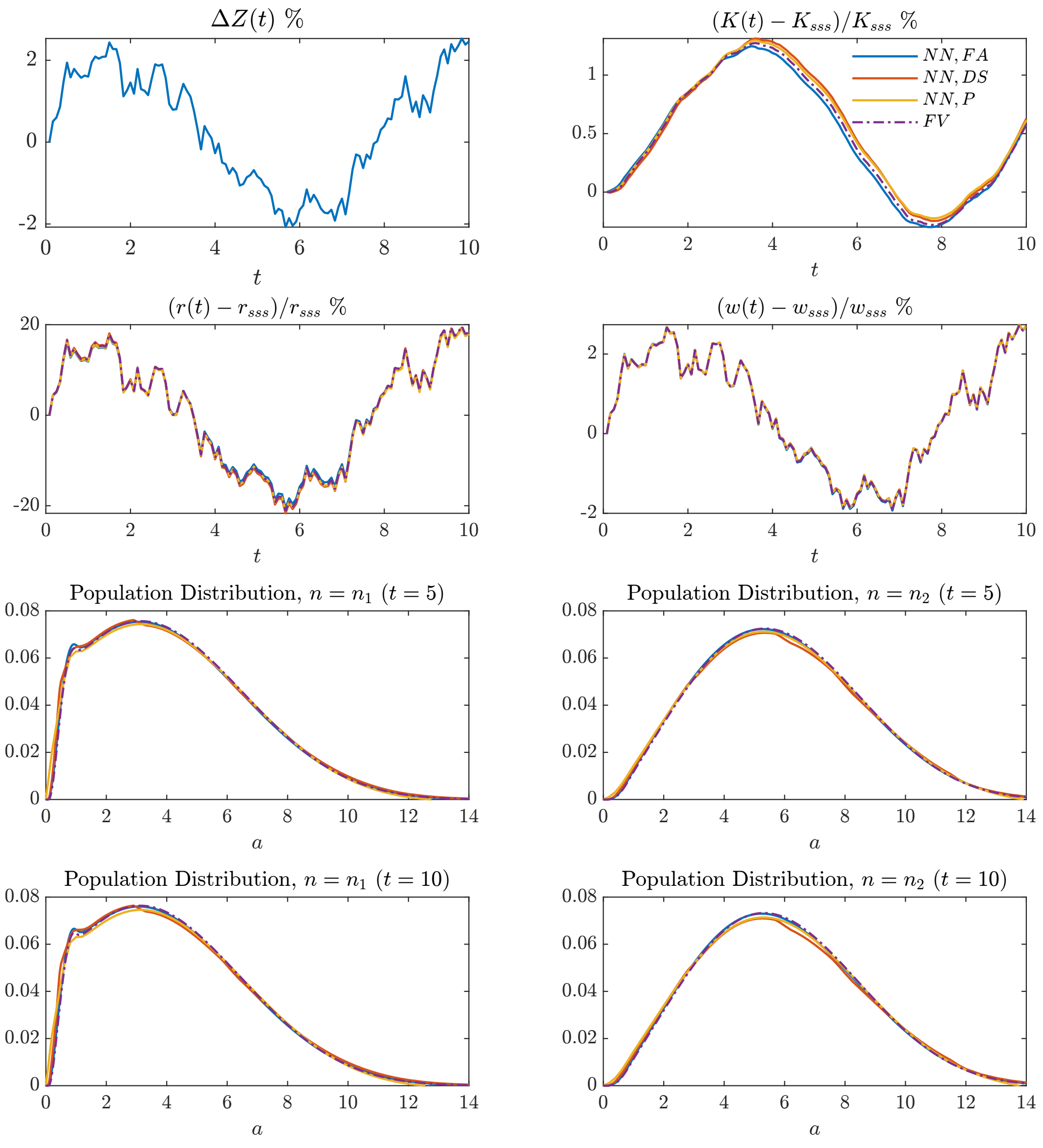

Figure 1: Simulations for the Krusell-Smith Model. The top left plot is the TFP shock path, the top right panel is the aggregate relative capital change. The second row left plot shows the relative change in the capital return and the second row right plot shows the relative change in the wage rate. The plots on the rows three and four show the distribution at different times in the simulation. *NN, FA* refers to the finite agent neural network, *NN, DS* refers to the discrete state space neural network, *NN, P* refers to the projection neural network, and *FV* refers to the result generated from Fernández-Villaverde et al. (2023). Subscript *sss* refers to the stochastic steady state at $z = 0$.

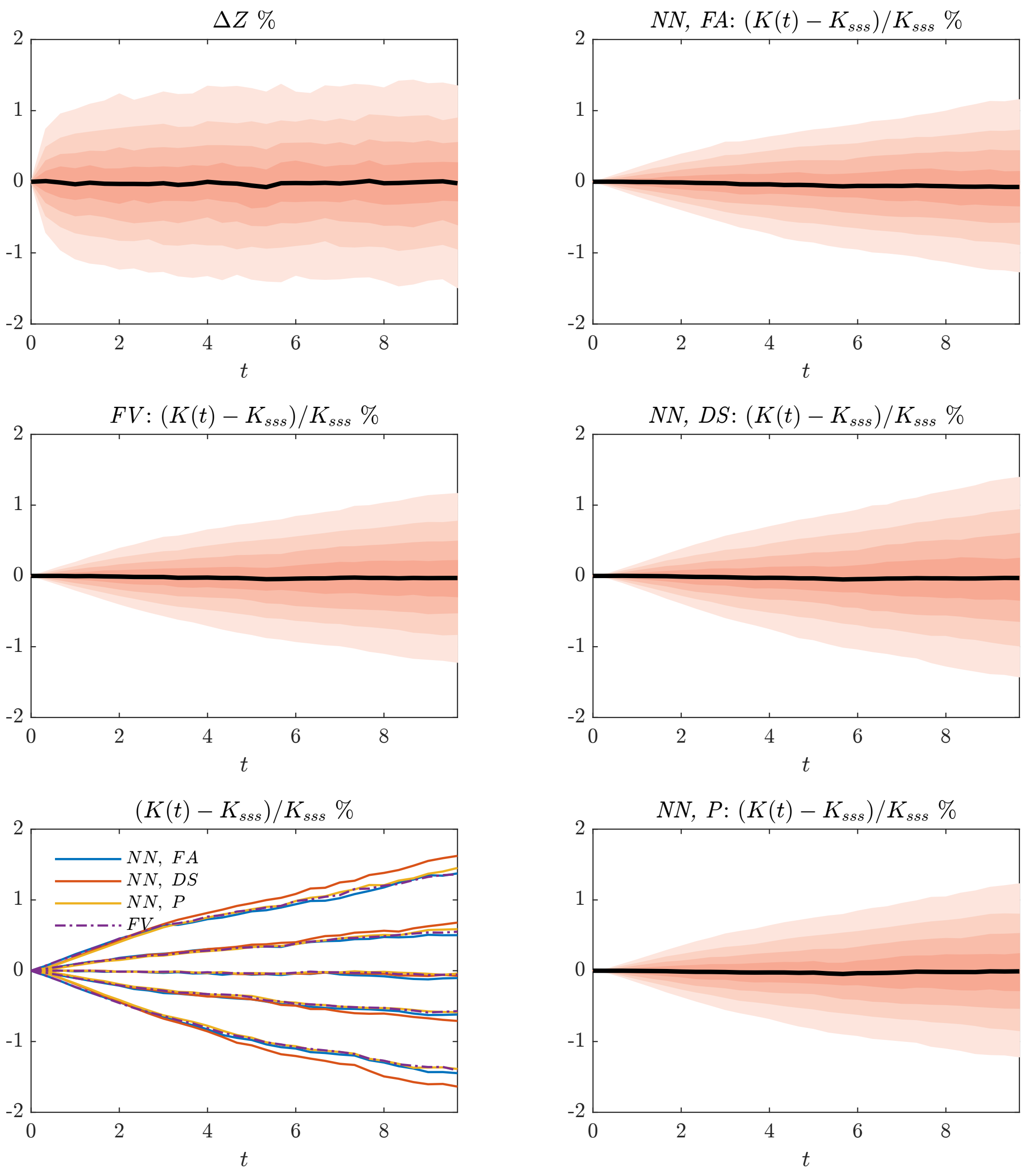

Figure 2: Forcasted aggregate capital dynamics starting from the stochastic steady state (*sss*) for the Krusell-Smith Model. The top left plot is the fan chart for the TFP shock path, generated from OU process with initial condition $z_0 = 0$. The middle left panel and all right panels are fan charts (capital quantile) of corresponding responses generated by Fernández-Villaverde et al. (2023) and the three neural network techniques. The bottom left panel is the time series plot for relative change in aggregate capital at quantile 10%, 30%, 50%, 70%, 90% (from the lowest to the highest), in which the solid lines are generated by neural network solutions and the purple dashed lines are generated by Fernández-Villaverde et al. (2023). *NN, FA*, *NN, DS*, and *NN, P* refer to the neural network technique based on finite agents, discrete state, and projection, respectively. *FV* refers to Fernández-Villaverde et al. (2023)'s technique.

Algorithm 2: Finding Transition Path by Neural Network: Krusell Smith case

```
Input  : Neural network approximations to the consumption rule $\hat{c}$, and pricing
         functions $(\hat{r}, \hat{w})$, number of agents $N$, time step size $\Delta t$, number of time
         steps $N_T$, number of simulations $N_{sim}$
Output: A transition path $g = \{g_t : t = 0, \Delta t, \dots, N_T \Delta t\}$

for $n = 0, \dots, N_T - 1$ do
    for $k = 1, \dots, N_{sim}$ do
        Sample $\Delta B_t^0$ from normal distribution $N(0, \Delta t)$, construct TFP shock path
          by: $z_{t+\Delta t} = z_t + \eta(\bar{z} - z_t) + \sigma \Delta B_t^0$.
        Sample $N$ agents $\{s_i : i = 1, \dots, N\}$ from distribution $g_t$ at time $t = n\Delta t$.
        Given other agents' state $s_{-i}$, calculate the consumption $\hat{c}(a, y, s_{-i}, z_{t+\Delta t})$,
          equilibrium capital return $\hat{r}(s_{-i}, z_{t+\Delta t})$ and wage $\hat{w}(s_{-i}, z_{t+\Delta t})$. Then
          construct $\mathcal{A}$ by finite difference scheme.
    end
    Take the average: $\bar{\mathcal{A}} = \frac{1}{N_{sim}} \sum_{k=1}^{N_{sim}} \mathcal{A}_k$
    Update $g_t$ by implicit method: $g_{t+\Delta t} = (I - \bar{\mathcal{A}}^\top \Delta t)^{-1} g_t$
end
```

5.4 Results With Fixed Aggregate Productivity

For fixed aggregate productivity, $z_t = z$, our example model is the continuous time version of the Aiyagari-Bewley-Huggett model discussed in Achdou et al. (2022). This model has a precise finite difference solution and so acts as a more detailed "check" for the accuracy of our solution technique. Table 4 plots the MSE between our steady state solution and the finite difference solution. Figure 3 plots the steady state consumption policy rule, derivative of the value function, probability density function (pdf), and cumulative distribution function (cdf) for the solutions from the finite agent code, the discrete state space code, and finite difference code. Evidently, the neural network solutions align very closely to the finite difference solution.

	Master equation loss	MSE(NN, FD)
Finite Agent NN	3.135×10^{-5}	4.758×10^{-5}
Discrete State Space NN	9.303×10^{-6}	6.591×10^{-5}

Table 4: Neural Networks' results for solving master equations. The master equation loss is the mean squared error of residuals. MSE(NN,FD) is the mean squared difference of consumption solved by neural network and finite difference. The training loss at each iteration is available in Appendix D.

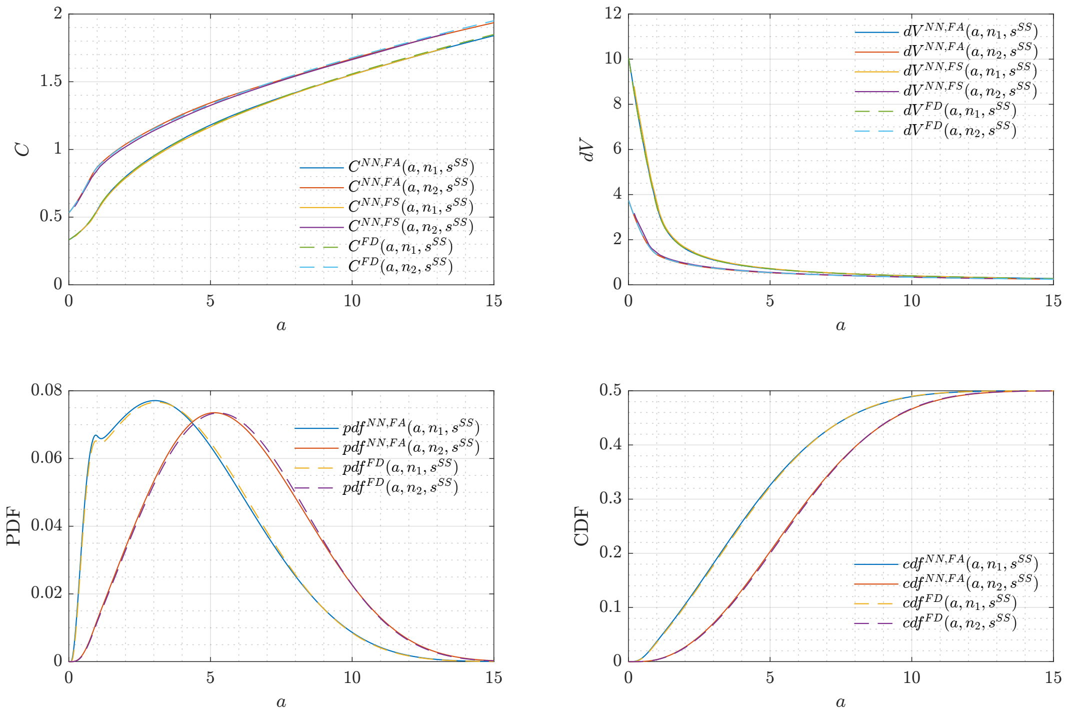

Figure 3: Comparison between neural network and finite difference solutions for the Aiyagari model. The top left plot shows the consumption policy rule. The top right shows the derivative of the value function, the bottom left shows the pdf, and the bottom right shows the cdf. The superscript NN, FA refers to the finite agent neural network code, the superscript NN, FS refers to the discrete state space neural network code, and the superscript FD refers to the finite difference code.

5.5 Transition Dynamics for MIT Shocks

Finally, we consider the transition path following an unexpected shock to aggregate productivity (a so-called "MIT" shock). Attempting to consider this makes little sense for the discrete state space approximation and the projection method because both rely on types of ergodic sampling and so have difficulty with attempting to consider unanticipated shocks. However, we were able to train our finite agent approximation using a moment sampling procedure that considered a large range of distributions and so it makes sense to consider how successfully the neural network approximation can handle transition paths.

An important advantage of having a global solution, i.e., c^* as a function of the distribution, is that we can solve for the transition path without a "shooting algorithm", as is commonly done in the finite difference literature. This is an advantage because shooting algorithms are often unstable, particularly for systems with a large number of prices that require complicated guess for the price path. Instead, with a full solution to the master equation, we can solve the Kolmogorov Forward Equation (KFE) directly as time-dependent consumption is a function of density $g(a,n)$.

$$\frac{\partial g_t(a,n)}{\partial t} = -\frac{\partial}{\partial a}(s(a,n,g_t)g_t(a,n)) + \lambda(g_t(a,\tilde{n}) - g_t(a,n)) \equiv \mathcal{A}^\top g$$

where $\mathcal{A}$ is transition matrix. We illustrate this for the finite agent approximation. The iterative procedure for solving the KFE in this case is summarized in Algorithm 3 below. We expand on the different approaches for calculating the transition path in Appendix B.2.

Algorithm 3: Finding Transition Path by Neural Network: Aiyagari case

```
Input : Neural network approximations for $(\hat{c},\hat{r},\hat{w})$, number of agents $N$, time
        step size $\Delta t$, number of time steps $N_T$, number of simulations $N_{sim}$
Output: A transition path $g = \{g_t : t = 0, \Delta t, \ldots, N_T\Delta t\}$

for $n = 0,\ldots,N_T - 1$ do
    for $k = 1,\ldots,N_{sim}$ do
        Sample $N$ agents $\{s_i : i = 1,\ldots,N\}$ from distribution $g_t$ at time $t = n\Delta t$.
        Given other agents' state $s_{-i}$, calculate the consumption $\hat{c}(a,n,s_{-i})$,
         equilibrium capital return $\hat{r}(s_{-i})$ and wage $\hat{w}(s_{-i})$. Then construct $\mathcal{A}_k$ by
         finite difference scheme.
    end
    Take the average: $\bar{\mathcal{A}} = \frac{1}{N_{sim}}\sum_{k=1}^{N_{sim}}\mathcal{A}_k$.
    Update $g_t$ by implicit method: $g_{t+\Delta t} = (I - \bar{\mathcal{A}}^\top\Delta t)^{-1}g_t$.
end
```

We compare the neural network and finite difference transition paths in Figure 4 below. We discuss how the finite difference solution is computed in Appendix B.2. In this numerical experiment, we train our neural network at $z = 0$ and we start from an economy in its steady state with productivity $z_t = -0.10$ for $t = 0$. At $t = 0^+$, an unexpected positive productivity shock brings z from -0.10 to $z_t = 0$ permanently. We solve distributional dynamics by Algorithm 3, and use steady state at $z = -0.10$ as the initial condition. We plot the percentage change of capital, capital return and wage evolution respectively in the first row and the second row of Figure 4. The difference between neural net's transition paths and finite difference's transition paths are less than 0.1%. The lower panels of Figure 4 compares neural network and finite difference probability density at time $t = 15$ and $t = 30$.

5.6 Calibration

In this section, we discuss how to use our algorithm to calibrate models. As has been discussed by a number of papers, a potential benefit of deep learning algorithms is that we can include the parameters, ζ, as additional inputs into the neural network:

$$V(\hat{X},\zeta) \approx \hat{V}(\hat{X},\zeta;\theta)$$

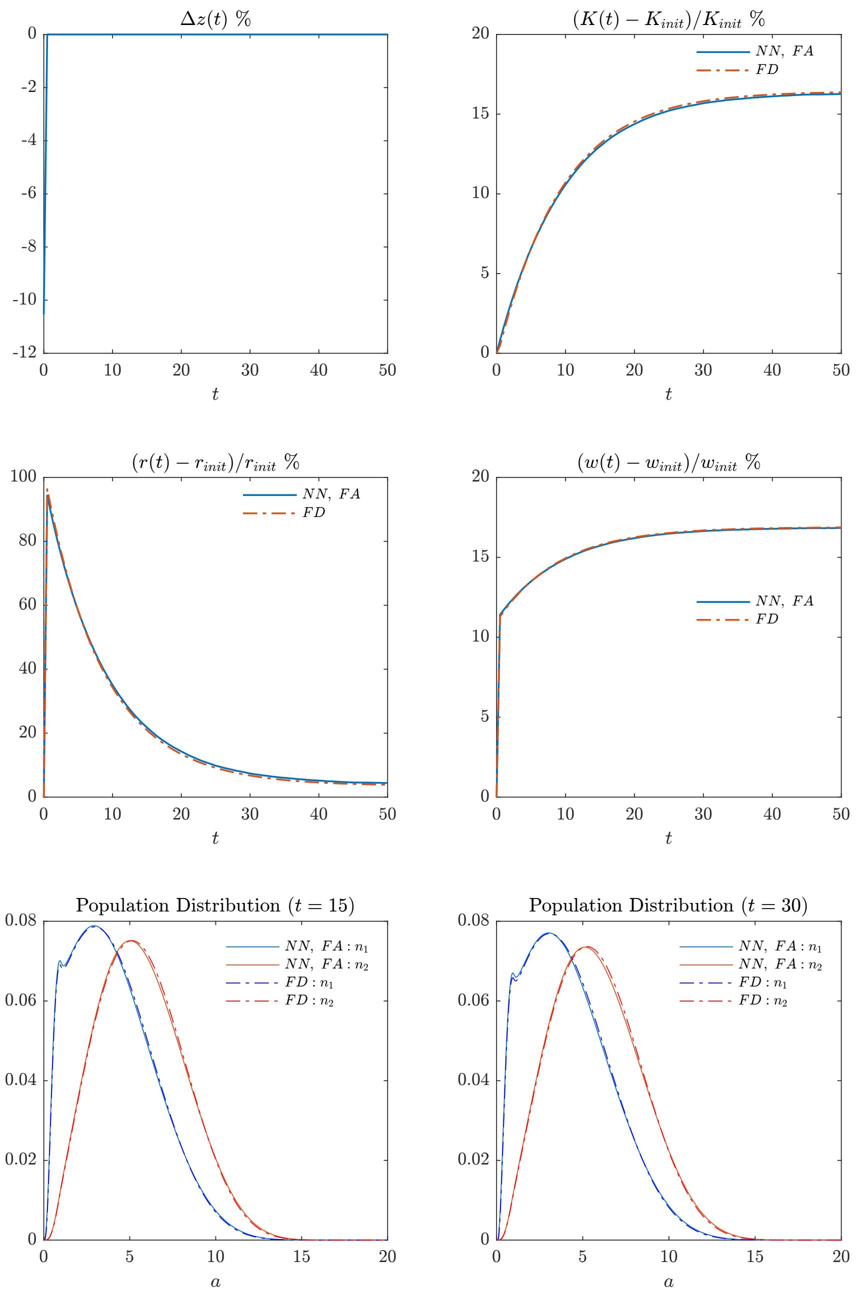

Figure 4: Comparison between neural network and finite impulse response for the Aiyagari model. The top left plot is the TFP shock path, the top right panel is the aggregate relative capital change, the middle left panel plots the relative capital return change, and the middle right panel plots the relative wage change. The bottom left and bottom right are snapshots of probability density at $t = 15$ and $t = 30$. *NN, FA* refers to the finite agent neural network code, and the *FD* refers to the finite difference code. Subscript *init* is the initial value at the steady state $z = z(0)$.

We can then train the neural network using sampling from both $\hat{X}$ and ζ. In principle, this means that we can train the model once and get a solution across a range of parameter values. We can then use $\hat{V}$ to calculate the moments for different parameters and calibrate the model. This approach is feasible if the neural network can be trained without ergodic sampling but difficult to implement if ergodic sampling is required.

We illustrate this technique for the finite agent approximation by calibrating the Krusell and Smith (1998) model to match a stochastic steady state capital-to-labor ratio of 5.0. We do this by including the borrowing constraint $\underline{a}$ as an input into the neural network, training the model randomly sampling $\underline{a}$ from $[-0, 1.5]$, and then using the trained model to solve for $\underline{a}$ that generates a stochastic steady state capital-to-labor ratio of 5.0. We show the results in Table 5.

K/L		
Target	Model	$\underline{a}$
5.0	5.0	1.082

Table 5: Neural Networks' results for Calibration

5.7 Comparison of Techniques

Although the different distribution approximation approaches can all be effective, we found they had different strengths and weaknesses in our experiments. We found the finite-agent approximation to be very robust in a number of ways: the neural network can be trained with a simple sampling procedure that does not require ergodic simulation (or any knowledge of the model solution), the algorithm only required a moderate number of agents (approximately 40), and we had success adding parameters as auxiliary states so the model could be solved across the state and parameter space at the same time.

By contrast, we found the discrete-state approximation to be difficult to work with for the Krusell and Smith (1998) model. We believe a lot of the issues come from having to approximate the derivatives in the Kolmogorov Forward Equation on the discrete state space. One challenge is that this requires a fine grid for agent wealth (approximately 200 grid points). A related challenge is that the training samples need to come from relatively smooth densities and so ergodic sampling is very important. Ultimately, this made the model slow to train. Follow up work in Payne et al. (2024) extends our approach to search and matching models and finds that the discrete state approximation is very effective for that class of models. This offers suggestive evidence that the discrete state approximation is most helpful when the KFE does not contain complicated derivatives (or potentially when it is trained by simulating a set of Forward-Backwards differential equations as in Huang et al. (2018)).

Finally, the projection approach brings a different set of trade-offs. Both the finite agent and discrete state approximations feed relatively large state spaces into the neural network and then let the neural network work out how to structure the approximate value function. By contrast, the projection method requires more choices ex-ante. This allows us to work with a much lower dimensional approximation (approximately 5 basis functions) and to choose which statistics of the distribution we want to match most closely in our approximation.

6 Lessons For Practitioners

Although the deep learning algorithm is straightforward to describe, we find that implementing it successfully can be tricky. In this section, we collect some lessons that we believe are helpful for using deep learning to solve macroeconomic models.

Lesson 1: Working out the correct sampling approach is very important. A key feature of continuous time methods is that we must specify where in the state space to sample. This is very different from discrete time approaches which typically simulate the economy and so implicitly sample from the ergodic distribution. Like Gopalakrishna (2021), we find that choosing where to sample is both an advantage (because we can focus sampling in interesting, rarely visited regions of the state space with complicated curvature) but also a difficulty (because it can be hard to know where to sample). Many of our initial problems were resolved by adjusting how we were sampling the finite dimensional approximation to the distribution. In particular, we found it is essential to have significant variation in distribution so that the neural network can learn where there is curvature in the problem.

Lesson 2: Neural networks have difficulty dealing with inequality constraints. The Achdou et al. (2022) formulation of the Aiyagari (1994) model is written with a hard lower boundary that $a_t \geq \underline{a}$, which leads to an inequality boundary condition at $\underline{a}$ and a mass point in the distribution at $\underline{a}$. This is convenient for the finite difference solution technique because the inequality constraint does not impact the tractability of the upwind scheme. However, it causes problems for the neural network approach because it is treated using a penalty term in the loss function, but the inequality constraint is too "easy" to satisfy, in the sense that the neural network will easily make the penalty term 0: many levels of the value function satisfy the constraint. Placing a low weight on the lower boundary error leads to solutions with the correct curvature but the wrong level. Placing a high weight on the lower boundary error leads to solutions with the right level but inaccurate curvature. So, there seems to only be a very small subset of weights that lead to accurate results for inequality boundary conditions. Ultimately, we find that replacing the hard constraint by a soft constraint and utility penalty makes the method much more robust while not changing drastically the model.

Lesson 3: Enforcing shape constraints is very important. We find that the neural network is likely to converge to "cheat solutions" that approximately solve the differential equation by setting derivatives to zero. There are number of ways to help with this, such as: (i) including in the loss function some terms on the curvature of the value function to enforce the correct shape (e.g. penalizing non-monotonicity or non-concavity), (ii) pre-training the neural network to match an initial guess satisfying known properties of the value function so that when the neural network is trained to minimize the PDE loss, it converges to a (local) minimizer which has the same desired properties, (iii) sampling from a sufficiently large part of the state space that the neural network realizes there is curvature in all dimensions, and (iv) choose an architecture which satisfies the constraints or which at least makes it easier for the neural network to satisfy these constraints. Overall, we did not observe that a single approach was sufficient by itself, but using a combination of these ideas helped.

Lesson 4: Mean squared errors can be misleading. We found that even if mean squared training errors are in the order of 10^{-2} or 10^{-3}, the neural network can give policy rules that are inaccurate. This suggests that overfitting can be a problem for neural network solutions to PDEs and so choosing the right "cross validation" sample is important. It also suggests that the threshold for convergence for neural network solutions to continuous time models might need to be higher than for other techniques, such as finite difference methods. The underlying question is related a posteriori error bounds, which would estimate how close to the true solution the neural network is in terms of the value of the loss.

Lesson 5: Start with a simple model to tune hyperparameters. A benefit and cost with using neural networks is that they are very flexible approximations. This means that a difficult and time consuming part of training a neural network model is finding appropriate hyperparameters. We find that it is helpful to start with a simple model for which we know the solution (or for which we have a very good approximation, e.g. a version of the model without aggregate shocks, that can be solved with a finite difference scheme), and tune the hyperparameters to make sure that the neural network approximation closely matches the finite difference solution. We can then introduce aggregate shocks starting with a set of hyperparameters that are good at capturing many features of the distribution. In other words, we suggest learning on a simple version of the problem, for which the solution is already known and then learning on the more complicated problem, using similar hyperparameters.

7 Conclusion

This paper proposed a new global solution algorithm for continuous time heterogeneous agent economies with aggregate shocks. We demonstrated our algorithm by solving two canonical models in the macroeconomics literature: the Aiyagari (1994) model and the

Krusell and Smith (1998) model. Using our method on the first model, which can be solved using classical techniques, allows us to find tune hyperparameters, which can then be used on more complex models. One advantage of our method is that solving the second model with aggregate shocks is not much more difficult than solving the first problem without aggregate shocks. However, this is only the beginning of what is possible with this technique. Future work can deploy this solution technique to solve high dimensional economic models with non-linearities and aggregate shocks.

References

Achdou, Y., Han, J., Lasry, J.-M., Lions, P.-L., and Moll, B. (2022). Income and wealth distribution in macroeconomics: A continuous-time approach. *The Review of Economic Studies*, 89(1):45–86.

Ahn, S., Kaplan, G., Moll, B., Winberry, T., and Wolf, C. (2018). When inequality matters for macro and macro matters for inequality. *NBER macroeconomics annual*, 32(1):1–75.

Aiyagari, S. R. (1994). Uninsured idiosyncratic risk and aggregate saving. *The Quarterly Journal of Economics*, 109(3):659–684.

Al-Aradi, A., Correia, A., Jardim, G., de Freitas Naiff, D., and Saporito, Y. (2022). Extensions of the deep Galerkin method. *Applied Mathematics and Computation*, 430:127287.

Auclert, A., Bardóczy, B., Rognlie, M., and Straub, L. (2021). Using the sequence-space Jacobian to solve and estimate heterogeneous-agent models. *Econometrica*, 89(5):2375–2408.

Azinovic, M., Gaegauf, L., and Scheidegger, S. (2022). Deep equilibrium nets. *International Economic Review*, 63(4):1471–1525.

Bayraktar, E., Budhiraja, A., and Cohen, A. (2018). A numerical scheme for a mean field game in some queueing systems based on Markov chain approximation method. *SIAM Journal on Control and Optimization*, 56(6):4017–4044.

Bensoussan, A., Frehse, J., and Yam, S. C. P. (2015). The master equation in mean field theory. *Journal de Mathématiques Pures et Appliquées*, 103(6):1441–1474.

Bertucci, C. and Cecchin, A. (2022). Mean field games master equations: from discrete to continuous state space. *arXiv preprint arXiv:2207.03191*.

Bhandari, A., Bourany, T., Evans, D., and Golosov, M. (2023). A perturbational approach for approximating heterogeneous-agent models.

Bilal, A. (2021). Solving heterogeneous agent models with the master equation. Technical report, University of Chicago.

Bretscher, L., Fernández-Villaverde, J., and Scheidegger, S. (2022). Ricardian business cycles. *Available at SSRN.*

Brzoza-Brzezina, M., Kolasa, M., and Makarski, K. (2015). A penalty function approach to occasionally binding credit constraints. *Economic Modelling*, 51:315–327.

Cardaliaguet, P., Delarue, F., Lasry, J.-M., and Lions, P.-L. (2015). The master equation and the convergence problem in mean field games. *arXiv.*

Carmona, R. and Laurière, M. (2021). Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games I: the ergodic case. *SIAM Journal on Numerical Analysis*, 59(3):1455–1485.

Carmona, R. and Laurière, M. (2022a). Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games: II—the finite horizon case. *The Annals of Applied Probability*, 32(6):4065–4105.

Carmona, R. and Laurière, M. (2022b). Deep learning for mean field games and mean field control with applications to finance. *Machine Learning in Financial Markets: A guide to contemporary practises, editors: A. Capponi and C.-A. Lehalle, Cambridge University Press.*

Carmona, R., Laurière, M., and Tan, Z. (2019). Model-free mean-field reinforcement learning: mean-field MDP and mean-field Q-learning. *arXiv preprint arXiv:1910.12802.*

Cohen, A., Laurière, M., and Zell, E. (2024). Deep backward and galerkin methods for the finite state master equation. *arXiv preprint arXiv:2403.04975.*

Delarue, F., Lacker, D., and Ramanan, K. (2020). From the master equation to mean field game limit theory: Large deviations and concentration of measure. *Annals of Probability*, 48(1):211–263.

Den Haan, W. (1997). Solving Dynamic Models with Aggregrate Shocks and Heterogeneous Agents. *Macroeconomic Dynamics*, 1(2):355–386.

Duarte, V. (2018). Machine learning for continuous-time economics. *Available at SSRN 3012602.*

Fernández-Villaverde, J., Hurtado, S., and Nuno, G. (2023). Financial frictions and the wealth distribution. *Econometrica*, 91(3):869–901.

Fernandez-Villaverde, J., Nuno, G., Sorg-Langhans, G., and Vogler, M. (2020). Solving high-dimensional dynamic programming problems using deep learning. *Unpublished working paper*.

Fouque, J.-P. and Zhang, Z. (2020). Deep learning methods for mean field control problems with delay. *Frontiers in Applied Mathematics and Statistics*, 6:11.

Frikha, N., Germain, M., Laurière, M., Pham, H., and Song, X. (2023). Actor-critic learning for mean-field control in continuous time. *arXiv preprint arXiv:2303.06993*.

Germain, M., Laurière, M., Pham, H., and Warin, X. (2022a). DeepSets and their derivative networks for solving symmetric PDEs. *Journal of Scientific Computing*, 91(2):63.

Germain, M., Mikael, J., and Warin, X. (2022b). Numerical resolution of mckean-vlasov fbsdes using neural networks. *Methodology and Computing in Applied Probability*, pages 1–30.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Gopalakrishna, G. (2021). Aliens and continuous time economies. *Swiss Finance Institute Research Paper*, (21-34).

Gopalakrishna, G., Gu, Z., and Payne, J. (2024). Asset pricing, participation constraints, and inequality. *Princeton Working Paper*.

Gu, H., Guo, X., Wei, X., and Xu, R. (2021). Mean-field controls with Q-learning for cooperative MARL: convergence and complexity analysis. *SIAM Journal on Mathematics of Data Science*, 3(4):1168–1196.

Hadikhanloo, S. and Silva, F. J. (2019). Finite mean field games: fictitious play and convergence to a first order continuous mean field game. *Journal de Mathématiques Pures et Appliquées*, 132:369–397.

Han, J., Jentzen, A., and E, W. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510.

Han, J., Yang, Y., and E, W. (2021). DeepHAM: A global solution method for heterogeneous agent models with aggregate shocks. *arXiv preprint arXiv:2112.14377*.

Hu, R. and Lauriere, M. (2022). Recent developments in machine learning methods for stochastic control and games. *ssrn.4096569*.

Huang, J. (2022). A probabilistic solution to high-dimensional continuous-time macro-finance models. *Available at SSRN 4122454*.

Huang, K. X., Li, Z., and Sun, J. (2018). Bank Competition , Directed Search , and Loan Sales.

Kahou, M. E., Fernández-Villaverde, J., Perla, J., and Sood, A. (2021). Exploiting symmetry in high-dimensional dynamic programming. Technical report, National Bureau of Economic Research.

Krusell, P. and Smith, A. A. (1998). Income and Wealth Heterogeneity in the Macroeconomy. *Journal of Political Economy*, 106(5):867–896.

Lacker, D. (2020). On the convergence of closed-loop Nash equilibria to the mean field game limit. *Annals of applied probability: an official journal of the Institute of Mathematical Statistics*, 30(4):1693–1761.

Laurière, M. (2021). Numerical methods for mean field games and mean field type control. *Mean Field Games*, 78:221.

Li, J., Yue, J., Zhang, W., and Duan, W. (2022). The deep learning Galerkin method for the general stokes equations. *Journal of Scientific Computing*, 93(1):1–20.

Lions, P.-L. (2007-2011). Lectures at College de France.

Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E. (2021a). DeepXDE: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228.

Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., and Johnson, S. G. (2021b). Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132.

Maliar, L., Maliar, S., and Winant, P. (2021). Deep learning for solving dynamic economic models. *Journal of Monetary Economics*, 122:76–101.

Min, M. and Hu, R. (2021). Signatured deep fictitious play for mean field games with common noise. In *International Conference on Machine Learning*, pages 7736–7747. PMLR.

Payne, J., Rebei, A., and Yang, Y. (2024). Deep learning for search and matching models. *Princeton Working Paper*.

Perrin, S., Laurière, M., Pérolat, J., Élie, R., Geist, M., and Pietquin, O. (2022). Generalization in mean field games by learning master policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9413–9421.

Prohl, E. (2017). Discetizing the Infinite-Dimensional Space of Distributions to Approximate Markov Equilibria with Ex-Post Heterogeneity and Aggregate Risk.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2017). Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561.*

Reiter, M. (2002). Recursive computation of heterogeneous agent models. *manuscript, Universitat Pompeu Fabra*, (July):25–27.

Reiter, M. (2008). Solving heterogeneous-agent models by projection and perturbation. *Journal of Economic Dynamics and Control*, 33:649–665 Contents.

Reiter, M. (2010). Approximate and Almost-Exact Aggregation in Dynamic Stochastic Heterogeneous-Agent Models. Technical report, Vienna Institute for Advanced Studies.

Sauzet, M. (2021). Projection methods via neural networks for continuous-time models. *Available at SSRN 3981838.*

Schaab, A. (2020). Micro and macro uncertainty. *Available at SSRN 4099000.*

Sirignano, J. and Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364.

Sznitman, A.-S. (1991). Topics in propagation of chaos. *Lecture notes in mathematics*, pages 165–251.

Winberry, T. (2018). A method for solving and estimating heterogeneous agent macro models. *Quantitative Economics*, 9(3):1123–1151.

A Details on Numerical Examples From Section 5 (Online Appendix)

A.1 Parameters

Parameter	Symbol	Value
Capital share	α	1/3
Depreciation	δ	0.1
Risk aversion	γ	2.1
Discount rate	ρ	0.05
Mean TFP	$\overline{Z}$	0.00
Reversion rate	η	0.50
Volatility of TFP	σ	0.01
Transition rate (1 to 2)	λ_1	0.4
Transition rate (2 to 1)	λ_2	0.4
Low labor productivity	n_1	0.3
High labor productivity	n_2	$1 + \lambda_2/\lambda_1(1 - n_1)$
Borrowing constraint	$\underline{a}$	10^{-6}
Maximum of asset	$\overline{a}$	20.0
Penalty Function	$\psi(a)$	$-\frac{1}{2}\kappa(a - a_{lb})^2$
Penalty parameters	a_{lb}	1.0
Penalty parameters	κ	3.0
Drift in O-U Process	η	0.5
Volatility in O-U Process	σ	0.01
Maximum TFP	z_{max}	0.04
Minimum TFP	z_{min}	-0.04

Table 6: Parameters for Krusell-Smith Model from Section 5.3

A.2 Detail on the Master Equations

In this subsection of the Appendix, we describe the precise master equations that we use to train the neural network for each approach. Let $W(a, l, z, \hat{\varphi}) := \partial_a V(a, l, z, \hat{\varphi})$ denote the derivative of the value function with respect to a.

Finite agent approximation: In this case, we replace the distribution by the positions of the agents:

$$\hat{\varphi} = \{(a^i, l^i)\}_{i \le I}$$

where $I = 41$ agents. The master equation operators can be written as:

$$\begin{aligned}
(\hat{\mathcal{L}}^h W)((a^i,l^i),z,\hat{\varphi}) = \; & + (r(\hat{\varphi}) - \rho) W((a^i,l^i),z,\hat{\varphi}) + \psi_a(a^i) + s_i \partial_a W((a^i,l^i),z,\hat{\varphi}) \\
& + \lambda(y^i) \left(W((a^i,\tilde{l}^i),z,\hat{\varphi}) - W((a^i,l^i),z,\hat{\varphi}) \right) \\
& + \partial_Z W(a^i,l^i,z,\hat{\varphi}) \eta(\overline{z} - z) + \frac{1}{2}\sigma^2 \partial_{zz} W(a^i,l^i,z,\hat{\varphi}) \\
(\hat{\mathcal{L}}^g W)((a^i,l^i),z,\hat{\varphi}) = \; & + \sum_{j \ne i} s_j((a^j,l^j),z,\hat{\varphi}) \partial_{a^j} W((a^i,l^i),z,\hat{\varphi}) \\
& + \lambda_j(l^j) \left(W((a^i,l^i),\{(a^j,\tilde{l}^j),z,\hat{\varphi}^{-j}\}) - W((a^i,l^i),z,\hat{\varphi}) \right)
\end{aligned}$$

Discrete state space approximation: In this case, we replace the distribution by its values at the points $x_1, \dots, x_N$ of the grid. More specifically, we take $N = 1000$ points, and we consider a discretization $a_1 < \cdots < a_{500}$ of the a-axis. We then denote $x_1 = (a_1, l_1), \dots, x_{500} = (a_{500}, l_1)$, $x_{501} = (a_1, l_2), \dots, x_{1000} = (a_{500}, l_2) \in \mathbb{R}^2$. We use a uniform grid and denote $\Delta a = a_2 - a_1$. For simplicity of presentation, we will use the notations $\hat{\varphi}_{m,i} = \hat{\varphi}_{i \times m}$ and

$$s_{m,i}(z) := s(a_m, l_i, \hat{c}^*(a_m, l_i, z, \hat{\varphi}), r(\hat{\varphi}), w(\hat{\varphi})).$$

Recall that the dynamics of the discrete distribution takes the generic form (3.4). In our implementation, we use the finite difference scheme proposed by Achdou et al. (2022). The KFE is replaced by the following finite difference equation:

$$d\hat{\varphi}_{m,i,t} = \hat{\mu}_{\hat{\varphi},m,i}(z_t, \hat{\varphi}_t)dt, \qquad m = 1, \dots, N/2, i = 1, 2,$$

with

$$\hat{\mu}_{\hat{\varphi},m,i}(z,\hat{\varphi}) := -[D_a(\hat{\varphi}_{\cdot,i} s_{\cdot,i}(z))]_m - \lambda(l_i)\hat{\varphi}_{m,i} + \lambda(l_{3-i})\hat{\varphi}_{m,3-i} \tag{A.1}$$

where $l_{3-i} = \check{l}_i = l_2$ if $i = 1$ and l_1 if $i = 2$, and the first order derivative is approximated using an upwind scheme:

$$\partial_a \varphi(a_m, l_i) s_{m,i}(z) \approx [D_a(\hat{\varphi}_{\cdot,i} s_{\cdot,i}(z))]_m := \frac{\hat{\varphi}_{m+1,i} - \hat{\varphi}_{m,i}}{\Delta a}(s_{m,i}(z))^+ + \frac{\hat{\varphi}_{m,i} - \hat{\varphi}_{m-1,i}}{\Delta a}(s_{m,i}(z))^-$$

with $(s_{m,i}(z))^+$ and $(s_{m,i}(z))^-$ denoting respectively the positive part and the negative part of $s_{m,i}(z)$. Viewing $\hat{\varphi}$ as an N-dimensional vector and using the notation $n = i \times m$, $\hat{\mu}_{\hat{\varphi},m,i}(z,\hat{\varphi})$ defined in (A.1) can be written in matrix form as:

$$\hat{\mu}_{\hat{\varphi},n}(z,\hat{\varphi}) = [A^{\hat{\varphi}} \hat{\varphi}]_n$$

where $A^{\hat{\varphi}}$ is an $N \times N$ tridiagonal matrix.

The operator $\hat{\mathcal{L}}^h$ is defined, similarly to the finite-agent approximation, by:

$$\begin{aligned}(\hat{\mathcal{L}}^h W)(a,l,z,\hat{\varphi}) = \quad & (r(\hat{\varphi}) - \rho)W(a,l,z,\hat{\varphi}) + \psi_a(a) \\ & + s(a,l,\hat{c}^*(a,l,z,\hat{\varphi}), r(\hat{\varphi}), w(\hat{\varphi}))\partial_a W(a,l,z,\hat{\varphi}) \\ & + \lambda(l)\left(W(a,\tilde{l},z,\hat{\varphi}) - W(a,l,z,\hat{\varphi})\right) \\ & + \partial_z W(a,l,z,\hat{\varphi})\eta(\overline{z} - z) + \frac{1}{2}\sigma^2 \partial_{zz} W(a,l,z,\hat{\varphi}).\end{aligned}$$

The operator $\hat{\mathcal{L}}^g$ defined in (3.5) is, in this context, given by:

$$\begin{aligned}(\hat{\mathcal{L}}^g W)(a,l,z,\hat{\varphi}) &= \sum_{i=1,2}\sum_{m=1}^{N/2} \hat{\mu}_{\hat{\varphi},m,i}(z,\hat{\varphi})\,\partial_{\hat{\varphi}_{m,i}} W(a,l,z,\hat{\varphi}) \\ &= \sum_{n=1}^{N} \hat{\mu}_{\hat{\varphi},n}(z,\hat{\varphi})\,\partial_{\hat{\varphi}_n} W(a,l,z,\hat{\varphi}),\end{aligned}$$

where $\partial_{\hat{\varphi}_{m,i}} W = \partial_{\hat{\varphi}_n} W$ denotes the partial derivative of W with respect to the coordinate $(i \times m) = n$ of the N-dimensional vector $\hat{\varphi}$.

Projection approximation: In this case, we replace the distribution by projection coefficients $\hat{\varphi} \in \mathbb{R}^5$ onto a basis $b_0, b_1, ..., b_5$ of 6 basis functions. We choose the basis functions as follows:

(i) We start out by solving the steady-state model with finite difference methods on a grid of 101 equally spaced grid points in the a-dimension. This finite-difference solution yields a 202×202-matrix that serves as a finite-dimensional approximation to the steady-state KFE operator $\mathcal{L}^{KF,ss}$. In a first step, we construct the basis of eigenfunctions described in the main text and discussed in more detail in Appendix C. Practically, we approximate the eigenfunctions by eigenvectors of the finite difference KFE matrix. We pick a total of 7 eigenvectors, which results in a preliminary basis $\tilde{b}_0^0, \tilde{b}_1^0, \ldots, \tilde{b}_6^0$.

(ii) In a second step, we impose the restriction that the marginal distributions in the y-dimension are always in line with the ergodic distribution of the y_t^i process. This restriction assures that effective labor is constant over time. It also reduces the dimension of the basis by 1. After imposing the restriction, we are thus left with a reduced basis $\tilde{b}_0, \tilde{b}_1, \ldots, \tilde{b}_5$, where $\tilde{b}_0 = \tilde{b}_0^0$ and $\tilde{b}_1, \ldots, \tilde{b}_5$ are each linear combinations of the original eigenfunctions $\tilde{b}_1^0, \ldots, \tilde{b}_6^0$.

(iii) In a third step, we make a change of variables that rotates the basis $\tilde{b}_0, \tilde{b}_1, \ldots, \tilde{b}_5$ but leaves the set of densities that can be approximated unaffected. Specifically, we first find the representing vector in span$\{\tilde{b}_1, ..., \tilde{b}_5\}$ for the linear functional

$$K(g) := \int ag(a, y_1)da + \int ag(a, y_2)da,$$

call it b_1 (existence is ensured by the Riesz representation theorem). We then select four more vectors out of $\tilde{b}_1, ..., \tilde{b}_5$ such that together with b_1 we obtain again a basis of the space and then project those four vectors onto the orthogonal complement of b_1. Call the resulting vectors $b_2, ..., b_5$. Also define $b_0 := \tilde{b}_0$. For our projection of the distribution, we work with the resulting basis $b_0, ..., b_5$. This rotation helps us in sampling because the coefficient $\hat{\varphi}_1$ on b_1 fully controls the aggregate capital stock implied by a given distribution approximation vector $\hat{\varphi}$,[11] so that it is relatively straightforward to implement a variant of moment sampling.

For the law of motion of the distribution approximation $\hat{\varphi}$, we adapt the generic approach outlined in the main text as follows for the KS model and the specific basis chosen here. We proceed in two steps:

First, we determine the law of motion of the coefficient $\hat{\varphi}_1$ that governs the evolution of aggregate capital K_t, so that we match the law of motion of K_t exactly. In the generic approach outlined in the main text, this amounts to picking the test function $\phi(a, y) = a$. Because we can match this component perfectly, we do not need to minimize regression residuals but instead directly compute the forward evolution

$$\mu_K(z, \hat{\varphi}) = \left.\frac{dK_t}{dt}\right|_{z_t=z, g_t=\hat{G}(\hat{\varphi})} = \left(Y(\hat{\varphi}, z) - \sum_{i=1,2} \int \hat{c}^*(a, l_i)\hat{G}(\hat{\varphi})(a, l_i)da - \delta K(\hat{G}(\hat{\varphi})) \right) dt,$$

where the integrals $\int \hat{c}^*(a, l_i)\hat{G}(\hat{\varphi})(a, l_i)da$ are approximated using quadrature over the 101-point grid on which the basis functions are known. We put particular emphasis on the evolution of K_t because this statistic of the distribution is all that matters for prices.[12] This procedure translates into a law of motion for $\hat{\varphi}_1$ as follows:

$$\mu_{\hat{\varphi},1}(z, \hat{\varphi}) = \frac{\mu_K(z, \hat{\varphi})}{K(b_1)}$$

For the remaining components $\hat{\varphi}_2, ..., \hat{\varphi}_5$ of the $\hat{\varphi}$-vector we take a more agnostic approach that mirrors closely the discrete state approximation. Specifically, we first compute the finite difference approximation of the KFE as in the discrete state space approximation on the 101-point grid on which the basis functions are known. Call this $\mu_g^{DS}(z, \hat{g})$, where $\hat{g}$ is the density on the 101-point grid (which corresponds to the distribution approximation

[11]By construction, the basis vectors $b_2, ..., b_5$ are orthogonal to b_1. Because b_1 is the representing vector for the K-functional, $K(b_2) = \cdots = K(b_5) = 0$, so these components do not contribute to the mean of the distribution.

[12]However, other aspects of the distribution still matter for the evolution of K_t itself as can be readily observed from the presence of the integrals over the consumption functions.

for the discrete state method).[13] We then determine the drifts $\mu_{\hat{\varphi},2}(z,\hat{\varphi}),...,\mu_{\hat{\varphi},5}(z,\hat{\varphi})$ as the coefficients of a linear regression (orthogonal projection) of $\mu_g^{DS}(z,\hat{G}(\hat{\varphi}))$ on the basis vectors $b_2,...,b_5$.

The regression just described can be mapped into the generic approach from the main text by choosing the test functions

$$\phi_m = \begin{cases} \delta_{(a_m,l_1)}, & m \leq 100 \\ \delta_{(a_{m-100},l_2)}, & m \geq 101 \end{cases}, \qquad m = 0,1,\ldots,201,$$

where $\delta_{(a,y)}$ denotes the Dirac delta distribution at (a,y).

With these choices, the master equation operators can be written as:

$$\begin{aligned}(\hat{\mathcal{L}}^h W)(a,l,z,\hat{\varphi}) = \quad & (r(\hat{\varphi})-\rho)W(a,l,z,\hat{\varphi}) + \psi_a(a) \\ & + s(a,l,\hat{c}^*(a,l,z,\hat{\varphi}),r(\hat{\varphi}),w(\hat{\varphi}))\partial_a W(a,l,z,\hat{\varphi}) \\ & + \lambda(l)\left(W(a,\tilde{l},z,\hat{\varphi}) - W(a,l,z,\hat{\varphi})\right) \\ & + \partial_z W(a,l,z,\hat{\varphi})\eta(\overline{z}-z) + \frac{1}{2}\sigma^2\partial_{zz}W(a,l,z,\hat{\varphi}) \\ (\hat{\mathcal{L}}^g W)(a,l,z,\hat{\varphi}) = \; & \sum_{n=1}^{5}\mu_{\hat{\varphi},n}(z,\hat{\varphi})\,\partial_{\hat{\varphi}_n}W(a,l,z,\hat{\varphi})\end{aligned}$$

A.3 Implementation Details

A.3.1 Network Structure

Finite Agent Agent Approximation: We use a fully connected feed-forward neural network with 5 layers and 64 neurons per layer. We use a tanh activation function between layers and a soft-plus activation at the output level. We initialize the neural network so that $W(a,\cdot)$ has an exponential shape with negative exponent. This is done through a pre-training phase.

Discrete State Space Approximation: The neural network for approximating $W = \partial_a V$ combines three steps to map the input data $\hat{X} := \{x,z,\hat{\varphi}\}$ into an output $\hat{W}(\hat{X};\theta_W)$. We describe these steps separately:

In a first step, an "embedding" network transforms the component $\hat{\varphi}$ into a 10-dimensional output $\hat{\varphi}'$ by feeding it through a fully connected feed-forward network as described in Section 4.1. This embedding network has 2 layers and 128 neurons per layer. We use a tanh activation function in the hidden layers and the identity function in the output layer. Denote by θ_W^e the collection of parameters for this network.

[13]Specifically, this means that if, in the context of the description of the discrete state space approximation, $\hat{\varphi} = \hat{g}$, then $\mu_g^{DS}(z,\hat{g}) := \mu_{\hat{\varphi}}(z,\hat{\varphi})$. Note that there, $\hat{\varphi}$ describes the values of the density on the 101-point grid, so this definition makes sense. However, in the present context $\hat{\varphi}$ has a different meaning (coefficients in the projection), so that we use the notation $\mu_g^{DS}(z,\hat{g})$ to avoid any confusion.

In a second step, we apply a recurrent network as proposed by Sirignano and Spiliopoulos (2018) to the modified input data $\hat{X}' := \{x, z, \hat{\varphi}'\}$, which results from $\hat{X}$ by replacing the distribution approximation $\hat{\varphi}$ with the output $\hat{\varphi}'$ if the embedding network. Specifically, the structure of the Sirignano and Spiliopoulos (2018) network is as follows:

$$
\begin{aligned}
h^{(0)} &= \phi^{(0)}(W^{(0)}\hat{X}' + b^{(0)}) && \text{...Hidden layer 0}\\
f^{(1)} &= \phi^{(1)}\left(W^{f,(1)}h^{(0)} + U^{f,(1)}\hat{X}' + b^{f,(1)}\right) && \text{...Hidden layer 1}\\
g^{(1)} &= \phi^{(1)}\left(W^{g,(1)}h^{(0)} + U^{g,(1)}\hat{X}' + b^{g,(1)}\right) && \text{...Hidden layer 1}\\
r^{(1)} &= \phi^{(1)}\left(W^{r,(1)}h^{(0)} + U^{r,(1)}\hat{X}' + b^{r,(1)}\right) && \text{...Hidden layer 1}\\
s^{(1)} &= \phi^{(1)}\left(W^{s,(1)}(r^{(1)} \odot s^{(0)}) + U^{s,(1)}\hat{X}' + b^{s,(1)}\right) && \text{...Hidden layer 1}\\
h^{(1)} &= (1 - g^{(1)}) \odot s^{(1)} + f^{(1)} \cdot h^{(0)} && \text{...Hidden layer 1}\\
&\vdots\\
f^{(H)} &= \phi^{(H)}\left(W^{f,(H)}h^{(H-1)} + U^{f,(H)}\hat{X}' + b^{f,(H)}\right) && \text{...Hidden layer } H\\
g^{(H)} &= \phi^{(H)}\left(W^{g,(H)}h^{(H-1)} + U^{g,(H)}\hat{X}' + b^{g,(H)}\right) && \text{...Hidden layer } H\\
r^{(H)} &= \phi^{(H)}\left(W^{r,(H)}h^{(H-1)} + U^{r,(H)}\hat{X}' + b^{r,(H)}\right) && \text{...Hidden layer } H\\
s^{(H)} &= \phi^{(H)}\left(W^{s,(H)}(r^{(H)} \odot s^{(H-1)}) + U^{s,(H)}\hat{X}' + b^{s,(H)}\right) && \text{...Hidden layer } H\\
h^{(H)} &= (1 - g^{(H)}) \odot s^{(H)} + f^{(H)} \cdot h^{(H-1)} && \text{...Hidden layer } H\\
o &= W^{(H+1)}h^{(H)} + b^{(H+1)} && \text{...Output layer}\\
\hat{Y} &= \phi^{(H+1)}(o) && \text{...Output}
\end{aligned}
$$

Here, $\{h^{(i)}\}_{0\leq i\leq H}$ denote the $H+1$ hidden layers and $\{f^{(i)}, g^{(i)}, r^{(i)}, s^{(i)}\}_{1\leq i\leq H}$ are auxiliary variables required to compute the neuron value in each layer. $W^{(0)}$, $\{W^{f,(i)}, W^{g,(i)}, W^{r,(i)}, W^{s,(i)}\}_{1\leq i\leq H}$, and $\{U^{f,(i)}, U^{g,(i)}, U^{r,(i)}, U^{s,(i)}\}_{1\leq i\leq H}$ are the weight matrices of the network, whereas $b^{(0)}$ and $\{b^{f,(i)}, b^{g,(i)}, b^{r,(i)}, b^{s,(i)}\}_{1\leq i\leq H}$ are the biases. The operator $\odot$ denotes the element-wise product (Hadamard product) of vectors. We choose $H = 3$ and 100 neurons per layer, which means that the dimension of each of the vectors $h^{(i)}$, $f^{(i)}$, $g^{(i)}$, $r^{(i)}$, and $s^{(i)}$ is 100. We use a tanh activation function in the hidden layers and an elu activation function in the output layer to ensure positivity of the output. The trainable parameters θ^r_W of this network consists of the collection of all weights and biases of the network.

In a final third step, we transform the output $\hat{Y}$ from the recurrent network into the approximate value for W as follows:

$$\hat{W} = \hat{Y}(a_0 + a)^{-\tilde{\eta}}.$$

Here a_0 and $\tilde{\eta}$ are non-trainable parameters. The additional factor $(a_0 + a)^{-\tilde{\eta}}$ is motivated by the hyperbolic shape of the marginal value function. Its inclusion helps improving the overall accuracy of the approximation for a given neural network size. We make the selection $a_0 = 10$ and $\tilde{\eta} = 0.5$.

The trainable parameters of the network $\hat{W}$ consists of the collection $\theta_W = \{\theta_W^e, \theta_W^r\}$ of all trainable parameters from the first and second step. We initialize all bias parameters to zero and all weight parameters randomly according to a uniform Xavier initialization (Glorot and Bengio, 2010).

In addition to the neural network for W, we also introduce an auxiliary network for the consumption function, $\hat{c}(\hat{X}; \theta_c)$. The structure of $\hat{c}$ precisely mirrors that of the network for $\hat{W}$ (including the number of layers and neurons in each sub-network), except that we omit the third step. For initialization of the parameters θ_c, we follow the same approach as for θ_W.

Projection Approximation: We use the same network structure and parameter initialization as for the discrete state space approximation. The only difference is that we choose a smaller embedding network, both for approximating W and for approximating c: instead of 128 neurons per layer, the embedding network for this approximation has only 64 neurons per layer.

A.3.2 Sampling

Finite Agent Approximation: We sample points of the form $\{(a^i, n^i), \{a^j, n^j\}_{j\in(I-1)}\}$ on the interior of the state space. For the idiosyncratic variable, a^i, we sample using an active sampling technique similar to those developed by Gopalakrishna (2021) and Lu et al. (2021a). We divide the state space for $a \in [0, 20]$ into 16 intervals with equal width. For each iteration of the algorithm, we calculate the average loss in each region. We then add extra points to the regions with the largest loss. We use the active sampling approach because it allows us to actively learn where the algorithm is having trouble minimizing the loss function. The interval $[\underline{a}, \overline{a}]$ is evenly partitioned into 2^4 subintervals. We do active sampling after 2,000 of epochs to make it efficient.[14] We calculate the residual error in each subinterval, find where it is the largest and add $2^4, 2^3, 2^2$ points to it, its nearest, and second nearest neighboring subinvervals. For sampling the population of agents, $\{a^j, n^j\}_{j\in(I-1)}$, we generate a random interest rate, then generate a random distribution of agents and scale their individual wealth so the equilibrium interest rate is the randomly drawn interest rate. The interest rate r is drawn from a uniform distribution on $[r_{lb}, r_{rb}]$, with $r_{lb} = -0.05, r_{rb} = 0.05$. Thanks to the fact that we deal with the boundary condition through a penalty, we do not need to sample points on the boundary, nor to evaluate the error $\mathcal{E}^b$. This is quite robust to different model parameters. We sample aggregate variable z uniformly from interval $[z_{min}, z_{max}]$.[15]

[14]The loss has a steeper drop after we start active learning in the 2000th epoch, as shown in Figure **??**.

[15]Theoretically, it is possible that z_t becomes lower than z_{min} or greater than z_{max}. In practice, letting $|z_{min/max}| = 4\sigma$ is good enough to approximate the economic relevant region. We cannot replace the step of having $r \in [r_{lb}, r_{rb}]$ by having $z \in [z_{min}, z_{max}]$ to let the distribution move around because z's scaling effect is weaker. In fact z's impact on interest rate is $\Delta r \approx (r + \delta_K)\Delta z$, which can hardly let the population

Discrete State Space Approximation: We first sample points of the form $(a, \hat{\varphi}, z)$ randomly and then construct the full sample for points of the form $(a, l, \hat{\varphi}, z)$ by using each originally sampled point twice, once in combination with $l = l_1$ and once in combination with $l = l_2$. We sample the three dimensions of $(a, \hat{\varphi}, z)$ independently as follows:

- We sample a from a uniform distribution over the domain $[0, 20]$ (without active sampling).
- We sample z from a uniform distribution over the domain $[z_{min}, z_{max}]$ as for the finite agent method.
- For the sampling of $\hat{\varphi}$, we use a mixture of two sampling schemes, (i) a variant of mixed steady state sampling and (ii) ergodic sampling based on the current approximation for W. For sampling scheme (i), we use a "degenerate" mixture based on just a single steady-state solution (for $z = 0$). Denote by $\hat{g}^{ss}$ the vector of steady-state density values on the discrete state grid. We take as sample points $\hat{\varphi}(a_i, l_i) = \frac{\omega_i \hat{g}^{ss}(a_i, l_i)}{\sum_{j=1}^{N_x} \omega_j \hat{g}^{ss}(a_j, l_j)}$, where the ω_i are i.i.d. with uniform distribution over an interval of the form $[1 - d_g, 1 + d_g]$, with $d_g \in (0, 1)$. We gradually increase the proportion of the sample that is according to (ii) from 0% to 90% during training.

Projection Approximation We follow precisely the same sampling approach as in the discrete state space approximation, except for the distribution dimension $\hat{\varphi}$. We therefore only discuss the latter here. We sample the first component of the distribution, $\hat{\varphi}_1$, separately as this component exclusively controls the aggregate level of the capital stock (compare Appendix A.2). We sample aggregate capital K from a uniform distribution over $[0.9K^{ss}, 1.1K^{ss}]$, where K^{ss} is the steady-state level of capital in the model without common noise and $z = 0$, and we adjust $\hat{\varphi}_1$ to match the sampled capital stock values. We sample the remaining four components $\hat{\varphi}_2, ..., \hat{\varphi}_5$ by combining uniform sampling from a hypercube centered around zero and ergodic sampling. We gradually increase the proportion of ergodic sampling from 0% to 80% during training.

A.3.3 Loss Function

For all three approximation method we impose penalties to impose the shape constraints $\partial_a W < 0$ and $\partial_z W < 0$ by choosing the shape error as follows:

$$\mathcal{E}^s(\theta^n, S^n) := \frac{1}{|S^n|} \sum_{(a,l,z,\hat{\varphi}) \in S^n} \Big(|\max\{\partial_a \hat{W}(a, l, z, \hat{\varphi}; \theta), 0\}|^2 + |\max\{\partial_z \hat{W}(a, l, z, \hat{\varphi}; \theta), 0\}|^2 \Big)$$

We combine this the equation residual error $\mathcal{E}^e(\theta^n, S^n)$ for the total error $\mathcal{E}(\theta^n, S^n)$ as described in Algorithm 1. We choose as weights $\kappa^e = 100, \kappa^s = 1$ for the finite agent

distribution move around. See more discussions in Section 6.

method and $\kappa^e = \kappa^s = 1$ for the other two methods. Ultimately, we have found that the relative weights of the loss components only matter in early training when the shape constraints are occasionally violated. In late training, shape constraints are typically always satisfied, so that the weights κ^e, κ^s are irrelevant for training.

B Additional Details on Aiyagari Model (Online Appendix)

B.1 Penalty Function Approximation

In this section, we compare the finite difference solutions with the hard and soft boundary constraints. The comparisons are summarized in Figure 5.

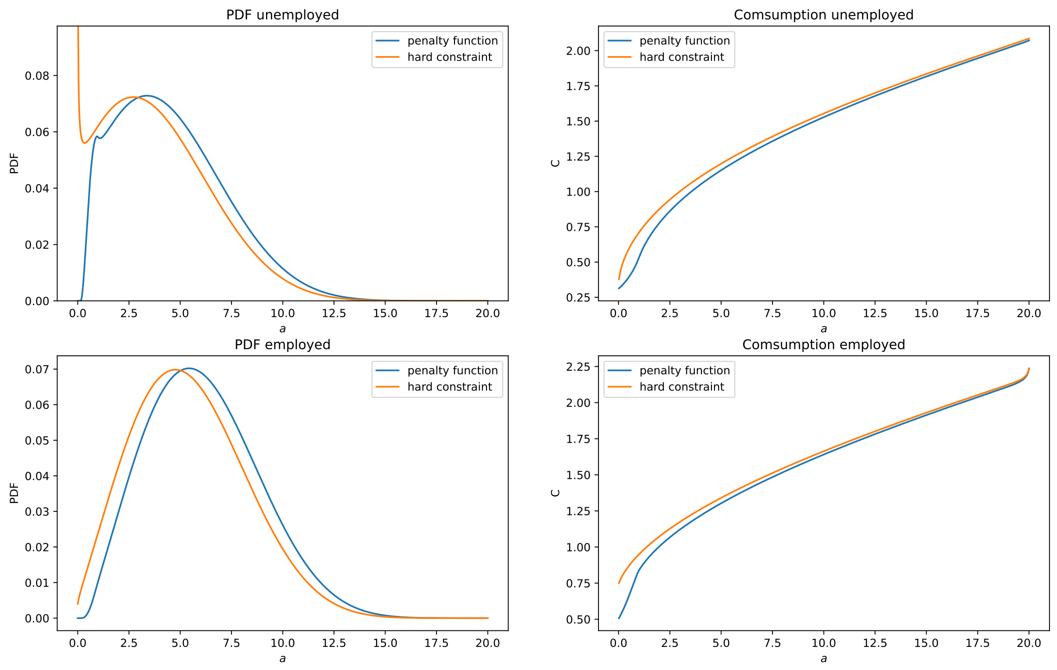

Figure 5: Comparison between penalty function approach and hard constraint in finite different exercise. $\psi(a) = -\frac{1}{2}\kappa(a - a_{lb})^2$, $a_{lb} = 1.0$, $\kappa = 3.0$.

B.2 Comparison of Techniques for Calculating Transition Dynamics without Aggregate Shock

After we obtained the neural network approximation of the value function in the master equation, $V(a_i, n_i, s_{-i})$, we are able to consider the transition path in two ways. The first way is to consider an evolving finite agent economy, the initial asset and employment status are sampled from the initial distribution: $g_0(a, n)$. This method gives the transition path of return on capital/labor directly. To solve the Kolmogorov Forward Equation, the key is to figure out $\mu(a, n) = ra + wn - c$, which can be approximated by the consumption policy. The algorithm can be summarized as follows in a high level.

Algorithm 4: Finding Transition Path by Neural Network

```
1. Sample $N$ agents from the initial condition $g_0(a, n)$.
2. Calculate the consumption, then find the transition path for every asset position.
3. After we've obtained $c_t, a_t, r_t$, solve $g_t(a, n)$ by finite difference of the forward
   equation
```

The second way is to update the distribution by Kolmogorov Forward Equation and resample agents from the updated distribution accordingly. Recall the forward equation in this economy:

$$\frac{\partial g_t(a, n)}{\partial t} = -\frac{\partial(\mu(a, n)g(a, n))}{\partial a} - \lambda_n g(a, n) + \lambda_{\check{n}} g(a, \check{n}).$$

In the this finite agents economy, the dynamic of equilibrium return on capital does not require another "guess-verify" loop, as in Achdou et al. (2022). The following algorithm modified to solve the dynamics of Aiyagari Economy.

Algorithm 5: Finding Transition Path by Finite Difference

```
1. Guess the path of equilibrium interest rate $r^o(t)$, then solve HJB, with terminal
   condition: $v(a, n, T) = \bar{v}(a, n)$
2. Solve the policy function $c_t(a, n)$.
3. Solve the forward equation, with initial condition $g_0(a, n)$.
4. Calculate the capital held by the whole economy: $\int_{\underline{a}}^{\infty} \sum_{n \in \{0,1\}} a g_t(a, n) = K_t$, then
   calculate the implied interest rate by $r^n(t) = \partial_K F(K_t, L) - \delta$, for every $t \in [0, T]$.
5. Update the path to be $\lambda r^o(t) + (1 - \lambda) r^n(t)$, repeat 1-4 until $||r^o - r^n||_\infty < \epsilon$.
```

Lessons. When using iterative method to find steady state/distributional dynamics in the forward equation, we have:

$$g = (I - \mathcal{A}(g)dt)^{-1} g,$$

which means we essentially have a high dimensional *fixed-point* problem. As we are using simulation to find $\mathcal{A}$, we know $\mathcal{A}$ may be very kinky because of the sampling error, and the

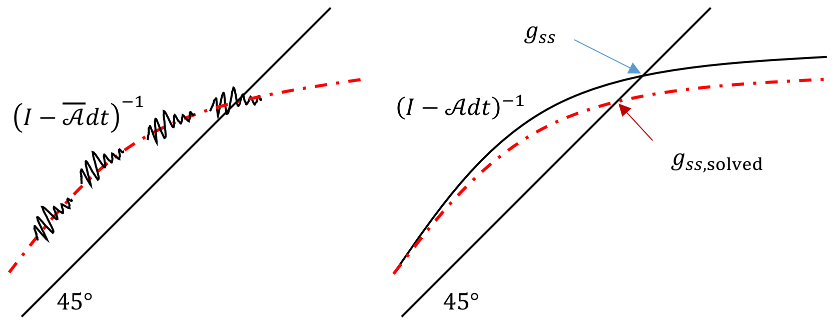

Figure 6: Fixed point problem in finding steady state and distributional dynamics. The fixed point is solved by the cross between $(I - \mathcal{A}dt)^{-1}$ and 45° line. The kinky black line on the left panel is the simulated operator $(I - \bar{\mathcal{A}}dt)^{-1}$, and the red dashed line is the smoothed approximation of $(I - \bar{\mathcal{A}}dt)^{-1}$. The concave black solid line on the right panel is the theoretical $(I - \bar{\mathcal{A}}dt)^{-1}$. g_{ss} and $g_{ss,\text{solved}}$ are the crossing points between the black solid line and 45° line, the red dashed line and 45° line, respectively.

solution is to have N_{sim} very large to smooth $(I - \bar{\mathcal{A}}dt)^{-1}$. Otherwise, the distribution g can be trapped before it reaches the steady state, as shown in Figure 6. Initial distribution with total capital $K_{init} < K_{ss,new}$ will end up with a oscillating, but a steady state with slightly lower aggregate capital. Similarly, initial distribution with total capital $K_{init} > K_{ss,new}$will end up with a oscillating, but a steady state with a slightly higher aggregate capital.

C Additional Details on the Eigenfunction Basis for the Projection Technique

We would like to choose a basis such that just a few basis functions are enough to provide a good approximation. The key idea to achieve this is to track the slow-moving or persistent dimensions of g_t while neglecting those dimensions that mean-revert fast. To understand why, recall that the only reason the distribution appears in the state space is because it helps agents forecast future prices q_t. Components that mean-revert fast carry only little information about future prices beyond a short time horizon. Neglecting them induces a comparably small error into the agent's forecasts.

The persistent dimensions of the distribution are related to certain eigenfunctions of the differential operator characterizing the KFE (2.6). We can rewrite this equation as

$$dg_t(x) = (\mathcal{L}_t^{KF} g_t)(x)dt$$

where, for generic distribution f and point x,

$$\begin{aligned}\mathcal{L}_t^{KF} f(x) = & -\partial_x[\mu^x(c_t^*(x, z_t, g_t), x, z_t, q_t) f(x)] + \frac{1}{2}\partial_{xx}[(\sigma^x(z_t))^2 f(x)] \\ & + \lambda((1-\gamma'(x, z_t, q_t)) f(x-\gamma(x, z_t, q_t)) - f(x)).\end{aligned}$$

Notice that $\mathcal{L}_t^{KF}$ is a linear differential operator that generally depends on time and is stochastic due to the implicit dependence on z_t and g_t. Constructing a (fixed) basis based on the eigenfunctions of $\mathcal{L}_t^g$ is therefore not directly possible because the set of eigenfunctions would itself be time-dependent and stochastic.

Instead, our construction is based on a related time-invariant operator $\mathcal{L}^{KF,ss}$. We define $\mathcal{L}^{KF,ss}$ in analogy to $\mathcal{L}_t^{KF}$ but for a perturbed model with common noise set to zero, $\sigma^z \equiv 0$, and under the assumption that the aggregate states $z_t = \bar{z}$ and $g_t = g^{ss}$ have reached a steady-state. There is a heuristic element in our basis construction that lies in the presumption that broadly the same dimensions of the distribution are relevant for the dynamics described by $\mathcal{L}^{KF,ss}$ and the true dynamics described by $\mathcal{L}_t^{KF}$. While generally plausible, if this requirement is not satisfied, then our proposed basis may not track the persistent dimensions of the true KFE dynamics well.

Let us consider the eigenfunctions $\{b_i : i \geq 0\}$ of $\mathcal{L}^{KF,ss}$ with corresponding eigenvalues denoted by $\{\lambda_i \in \mathbb{R} : i \geq 0\}$. They satisfy:

$$\mathcal{L}^{KF,ss} b_i = \lambda_i b_i, \qquad i \geq 0.$$

Suppose g^{ss} is the unique stationary distribution and the dynamics prescribed by the KFE are locally stable around g^{ss}. Then the eigenvalue $\lambda_0 = 0$ exists and its associated eigenfunction is, up to scaling, $b_0 = g^{ss}$. All remaining eigenvalues satisfy $\Re\lambda_i < 0$, so that these components of the distribution mean-revert to zero over time. Furthermore, a smaller $\Re\lambda_i$ is associated with a faster speed of mean-reversion. To be precise, suppose $g_t = g^{ss} + \sum_{i=1}^{\infty} \varphi_{i,t} b_i$ is the time-t distribution expressed as a linear combination of all the eigenfunctions and suppose the time evolution of g_t is described by the differential operator $\mathcal{L}^{KF,ss}$. Then,

$$g^{ss} + \sum_{i=1}^{\infty} \dot{\varphi}_{i,t} b_i = \frac{dg_t}{dt} = \mathcal{L}^{KF,ss} g_t = \mathcal{L}^{KF,ss} g^{ss} + \sum_{i=1}^{\infty} \varphi_{i,t} \mathcal{L}^{KF,ss} b_i = g^{ss} + \sum_{i=1}^{\infty} \varphi_{i,t} \lambda_i b_i$$

Since the b_i form a basis, the previous equation implies a system of ordinary differential equations for the coefficient functions $\varphi_{i,t}$, $i \geq 1$. The solution is given by $\varphi_{i,t} = \varphi_{i,0} e^{\lambda_i t}$, and therefore

$$g_t = g^{ss} + \sum_{i=1}^{\infty} \varphi_{i,0} e^{\lambda_i t} b_i, \qquad t \geq 0.$$

Because $\Re\lambda_i < 0$ for $i \geq 1$, all terms in the series on the right decay to zero as $t \to \infty$. Components corresponding to eigenvalues λ_i with very negative real parts decay at a faster

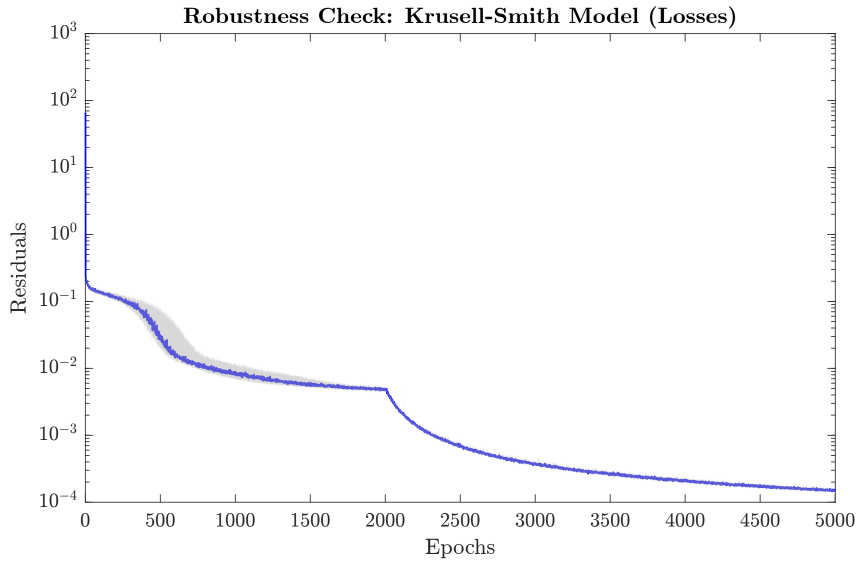

Figure 7: Training Loss vs Interation Plots (Finite Agent Method). Shaded areas are 80% confidence interval.

rate. As such, deviations from the stationary distribution g^{ss} have a greater persistence if they are in the direction of eigenfunctions corresponding to eigenvalues that are (negative but) close to 0. We therefore expect that a finite basis of eigenfunctions will provide a good approximation of the infinite sum if there are only a few “significant” eigenvalues that are close to 0.

The previous considerations motivate our basis choice in the main text. To be precise, that basis choice means the following: we order with descending real parts, $\lambda_0 = 0 > \Re\lambda_1 > \Re\lambda_2 > \dots$, and choose $b_0, b_1, ..., b_N$ as the eigenfunctions corresponding to the first $N+1$ eigenvalues in this ordering as our basis. We remark here also that this choice of $b_0, b_1, ..., b_N$ satisfies the required properties stated in equation (3.6) of Section 3.3. This is clear for $n = 0$. For $n > 0$, it follows form the fact that the operator $\mathcal{L}^{KF,ss}$ describes a mass-preserving evolution, which is only consistent with asymptotic decay over time if the integral is zero.

D Training Losses (Online Appendix)

Figure 7 show the distribution of training paths across 20 runs of the algorithm.