Deep Learning Solutions of Large Non-Convex Life-Cycle Models[*]

Preliminary

Jeppe Druedahl[†], Raphaël Huleux[‡], Jacob Røpke[§]

February 27, 2026

Abstract

We introduce a novel deep learning algorithm for solving large life-cycle models with both continuous and discrete choices. This allows us to simultaneously account for both labor supply choices with human capital accumulation, portfolio choices with a risky and a risk-free asset, and housing and mortgage choices. We work directly on the Bellman equation and approximate both value and policy functions with neural networks, and use a simulated training sample instead of tensor product grids. This substantially alleviates the curse of dimensionality. We demonstrate this in a consumption-saving model with multiple durable goods subject to non-convex adjustment costs where our deep learning algorithm clearly outperforms a standard value function iteration. We confirm that we can accurately solve a large life-cycle model in 12 hours on a single GPU. We solve the model simultaneously across all periods instead of with backward induction. This simplifies transfer learning, where a new solution for a new set of parameters in a calibration or estimation can easily be achieved. An accompanying easy-to-use software package implements the method.

Python package: github.com/NumEconCopenhagen/EconDLSolvers

[*]We are grateful for helpful comments from Jonathan Parker, Yucheng Yang, Marlon Azinovic, Jan Žemlička, Pontus Rendahl, Jacob Adenbaum, Simon Haastert, Anders Munk-Nielsen, Thomas Høgholm Jørgensen, Bertel Schjerning, Adam Jørgensen and Annasofie Marckstrøm Olesen. We thank Martin Andreas Kildemark and Mikkel Østergaard Reich for excellent research assistance. Funding from Independent Research Fund Denmark (Grant 2065-00028B) and Riksbankens Jubileumsfond (Grant M23-0019) is gratefully acknowledged. Center for Economic Behavior and Inequality (CEBI) is a center of excellence at the University of Copenhagen, founded in September 2017, financed by a grant from the Danish National Research Foundation (Grant DNRF134). A subset of the results in this paper was previously contained in the paper »Deep Learning Algorithms for Solving Finite-Horizon Models« by Druedahl and Røpke (2024).

[†]CEBI, Department of Economics, University of Copenhagen. E-mail: jeppe.druedahl@econ.ku.dk.

[‡]CEBI, Department of Economics, University of Copenhagen. E-mail: rph@econ.ku.dk.

[§]CEBI, Department of Economics, University of Copenhagen. E-mail: jro@econ.ku.dk.

1 Introduction

We introduce a novel deep learning algorithm for solving large finite-horizon models. The algorithm allows for both continuous and discrete choices, and does not require first order conditions to be sufficient or choice sets to be convex. It can thus solve a much broader class of models compared to existing deep learning methods used in economics.

We focus on solving life-cycle models[1], and we show our algorithm makes it possible to accurately solve models simultaneously accounting for both labor supply choices with human capital accumulation, portfolio choices with a risky and a risk-free asset, and housing and mortgage choices. We do this in 12 hours on a single GPU. This implies that even larger models can also be solved. Our algorithm allows for transfer learning, where the solution at a previous parameter set can be used as the initial guess for a new parameter set, and for including heterogeneous parameters as surrogate states. Both simplify estimation. Together, our algorithm allows researchers to work with more realistic descriptions of the environment households make decisions in, which will ultimately improve the policy guidance the models can provide.

Our algorithm works directly on the Bellman equation, and we denote it DeepV. We approximate both the post-decision value function and the policy function for the continuous choices with deep neural networks. We can then derive the optimal discrete choices directly from these together with potential taste shocks. Neural networks can basically approximate any function (Hornik et al., 1989), and are in practice able to provide accurate approximation of high-dimensional functions with a computationally feasible number of parameters. We do not specify tensor product grids, but instead train the neural networks on a simulated training sample using a variant of stochastic gradient descent. In sum, this substantially alleviates the curse of dimensionality.

Our proposed algorithm can be seen as a form of value function iteration. Instead of using backward induction period-by-period, we solve it across all periods simul-

[1] The origin of life-cycle models can be traced back to at least Modigliani and Brumburg (1954). A seminal paper is Gourinchas and Parker (2002), which was the first estimation of the structural parameters of a life-cycle model with precautionary savings. Modern surveys of various aspects of the vast literature can be found in Heathcote et al. (2009); Nardi et al. (2016); Gomes (2020).

taneously. This is beneficial because value and policy functions share a lot of properties across periods, which then only needs to be learned once. We instead use a final backward iteration to verify that our policy function cannot be improved upon. Two details of our algorithm turn out to be important for precision: i) We allow for exploration in the simulation, which ensures that a broader set of states than those under the true policy function is visited and thus included in training. This follows the approach in Druedahl and Røpke (2026). ii) We use multiple neural networks for the value function and average them to reduce noise.

We furthermore show that targeting the necessary (but not sufficient) first order conditions when training the policy network yields faster convergence and additional accuracy. In our preferred network structure, the policy network simultaneously outputs the continuous choices across all potential discrete choices. This is done to avoid some discrete choices never being selected due to a poor approximation of the conditional continuous choices. The computational time cost loss is arguably limited when the number of discrete choices is not too large.

We first test our algorithm in a consumption-saving model with multiple durable goods subject to non-convex adjustment costs. This is a complicated model, but can still be solved with a standard value function iteration, which we compare our solution to. Already with two durable goods our algorithm finds a better solution measured in expected discounted utility than the standard function iteration. The solution time is also similar, but the hardware cost is larger for the deep learning algorithm. Euler-errors are slightly larger for the deep learning solution, but this disadvantage disappears when including information from the first order conditions in the deep learning solution. We find no material differences across economic outcomes. We provide numerical experiments showing the importance of exploration in the training samples and averaging across multiple value networks. We finally show that solution time is drastically reduced if we need to re-solve the model for a new set of parameters. A standard backward induction approach requires re-solving the model from scratch.

We then turn to an actual application of our algorithm solving a large life-cycle, where households make decisions regarding both their labor supply, consumption, housing, and financial portfolio of risky and safe assets. This model cannot be solved with conventional methods, but can still be solved with our deep learning algorithm in 12 hours on a single GPU. We use four approaches to confirm that, we have found the true solution: i) the average expected discounted utility stabilize over training, ii) no material further improvements are achieved by a backward induction step, iii) Euler-errors are satisfactory, iv) economic outcomes stabilize over training.

Related literature. The use of deep learning is rapidly expanding in economics. Fernández-Villaverde et al. (2024) starts from the basics explaining why this can help tame the curse of dimensionality. Most work has focused on general equilibrium models(e.g. Maliar et al., 2021; Azinovic et al., 2022; Azinovic and Žemlička, 2023; Gu et al., 2024; Payne et al., 2024; Kase et al., 2024; Azinovic-Yang and Žemlička, 2025; Han et al., 2025).

The literature on single agent models is smaller. Maliar et al. (2021) solve a simple infinite horizon consumption-saving model and Pascal (2024) extends this to a multi-dimensional exogenous income process developing a general bias-corrected Monte Carlo operator. We focus on large and complex finite-horizon models, but many of the insights from our solution algorithm can also be used in infinite horizon. Our previous work, Druedahl and Røpke (2026) considers large finite-horizon models with many states, choices and constraints, but restricts attention to convex models. For smaller models, we find that an algorithm called DeepFOC based on minimizing errors in first order conditions is the best. For larger models, we find that an algorithm based on simulation alone, called DeepSimulate, is preferable. We show in Appendix A.3 that the DeepV algorithm developed in this paper performs worse than DeepFOC for the canonical buffer-stock model. Both DeepFOC and DeepSimulate can, however, not be applied for models with discrete choices and DeepFOC require first order conditions to be both necessary and sufficient.

Maliar and Maliar (2022) and Duarte et al. (2022) are closest to this paper, and are the only other papers to consider discrete choices in connection with deep learning. Maliar and Maliar (2022) consider an indivisible labor choice in an otherwise simple household problem embedded in a general equilibrium model. Duarte et al. (2022) builds on Sehnke et al. (2010) and solves a complex life-cycle using a simulation based approach extended to allow for discrete choices by assuming all choices initially are probabilistic.

Alternative approaches for alleviating the curse of dimensionality is using (adaptive) sparse grids (Judd et al., 2017; Brumm and Scheidegger, 2017; Brumm et al., 2021) or Gaussian Processes (Scheidegger and Bilionis, 2019), but the former is challenging with irregular domains, and the latter does not handle large amounts of data efficiently (see Azinovic et al. (2022) for a further discussion of these trade-offs).

2 Solution Algorithm

In this section, we introduce a deep learning algorithm for solving a broad class of finite-horizon models with discrete and continuous choices. Our algorithm can be seen as a form of Value Function Iteration (VFI): We approximate the value and policy functions using neural networks, and instead of solving the maximization problem on a tensor product grid, we solve the model on an endogenous training sample simulated using the current policy and value functions. We solve the problem simultaneously across all periods, instead of using backward induction, which allows the network to learn the time-dependencies in the model directly.

Below we first explain the model class considered, and then turn to how the value and policy functions are approximated with neural networks, how we construct our training samples, and finally how the parameters of the neural networks are updated.

2.1 Model Class

Time is discrete and indexed by $t \in \{0, 1, \dots T-1\}$. The notation is as follows:

1. The *states* are denoted by s_t (continuous or discrete).
2. The *discrete choice* is denoted by $d_t \in \mathcal{D}_t(s_t) = \{d^0, d^1, \dots, d^{\#_d - 1}\}$.
3. The *taste shocks* are $\varepsilon_t^d = \{\varepsilon_t^{d,0}, \varepsilon_t^{d,1}, \dots, \varepsilon^{d,\#_d-1}\}$ with i.i.d. $\varepsilon_t^{d,j} \sim \text{Gumbel}(0, \sigma_\varepsilon)$.
4. The *continuous choices* are denoted by $a_t \in \mathcal{A}_t(s_t, d_t)$.
5. The *utility function* is $u_t(s_t, d_t, a_t)$.
6. The *post-decision states* are $\bar{s}_t = \overline{\Gamma}_t\left(s_t, d_t, a_t\right)$.
7. The *stochastic shocks* are z_{t+1}, drawn from the distribution $F_t^z(\bar{s}_t)$.
8. The *transition function* is $s_{t+1} = \Gamma_t\left(\bar{s}_t, z_{t+1}\right)$.
9. The *terminal post-decision value function* is $h(\bar{s}_{T-1})$.

This encapsulates a broad model class. In particular, we do not enforce any restrictions to ensure the first order conditions for the continuous choices are sufficient. For notational simplicity, we write it as if there are always $\#_d$ discrete choices.[2]

The timing assumption is as follows:

1. Taste shocks ε_t^d are realized.
2. Discrete choices d_t are made.
3. Continuous choices a_t are made.
4. Stochastic shocks z_{t+1} are realized.

The solution algorithm can also easily be extended to allow for more general timing assumptions, including shocks arriving between the discrete choice and the continuous choices.

The value-of-choice is

$$\begin{aligned} \mathcal{V}_t(s_t, d_t, a_t) &= u_t(s_t, d_t, a_t) + \beta \begin{cases} h(\overline{s}_t) & \text{if } t = T-1 \\ \overline{v}_t\left(\overline{s}_t\right) & \text{else} \end{cases} \\ \overline{s}_t &= \overline{\Gamma}_t\left(s_t, d_t, a_t\right) \end{aligned} \tag{1}$$

where$\beta > 0$ is the discount factor.[3] The optimal policy function for the continuous choices, conditional on the discrete choice, is then

$$\pi_t^a(s_t, d_t) = \arg\max\nolimits_{a_t \in \mathcal{A}_t(s_t, d_t)} \mathcal{V}_t(s_t, d_t, a_t), \tag{2}$$

and the optimal policy function for the discrete choice is

$$\pi_t^d(s_t, \varepsilon_t^d) = \arg\max\nolimits_{d_t \in \mathcal{D}_t(s_t)} \mathcal{V}_t(s_t, d_t, \pi_t^a(s_t, d_t)) + \varepsilon_t^d(d_t), \tag{3}$$

where $\varepsilon_t^d(d_t)$ is the effective taste shock,

$$\varepsilon_t^d(d_t) \equiv \sum_{j=0}^{\#_d - 1} \mathbf{1}_{d^j = d_t} \varepsilon_t^{d,j}$$

The Gumbel assumption implies that the post-decision value function can then be

[2] We can also allow the scale of the taste shocks to vary by time period and state, $\sigma_{\varepsilon,t}(s_t)$.

[3] We can allow the discount factor to be time and state dependent, $\beta_t(s_t)$, and our method can also be used with Epstein-Zin preferences.

written

$$\begin{aligned}\overline{v}\left(\overline{s}_t\right) &= \mathbb{E}_t^z\left[\mathbb{E}_t^{\varepsilon}\left[\max_{d_t\in\mathcal{D}_t(s_t)} \mathcal{V}(s_t,d_t,\pi_t^a(s_t,d_t)) + \varepsilon_t^d(d_t)\right]\right] \\ &= \mathbb{E}_t^z\left[\sigma_\varepsilon \log\left[\sum_{d_t\in\mathcal{D}_t(s_t)} \exp\left(\frac{\mathcal{V}(s_t,d_t,\pi_t^a(s_t,d_t))}{\sigma_\varepsilon}\right)\right]\right] \\ z_{t+1} &\sim F_t^z(\overline{s}_t) \\ s_{t+1} &= \Gamma_t\left(\overline{s}_t, z_{t+1}\right)\end{aligned} \tag{4}$$

The *expected discounted lifetime* utility given policiesπ^a and π^d is

$$\begin{aligned}R\left(\pi^a,\pi^d\right) &= \mathbb{E}_{-1}\left[\sum_{t=0}^{T-1}\beta^t\left(u_t(s_t,a_t,d_t)+\varepsilon_t^d(d_t)\right)+\beta^{T-1}h(\overline{s}_{T-1})\right] \\ s_0 &\sim F_0^s \\ \varepsilon_t^d &\sim \text{i.i.d. Gumbel}(0,\sigma_\varepsilon) \\ d_t &= \pi_t^d(s_t,\varepsilon_t^d) \\ a_t &= \pi_t^a(s_t,d_t) \\ \overline{s}_t &= \overline{\Gamma}_t(s_t,d_t,a_t) \\ z_{t+1} &\sim F_t^z(\overline{s}_t), \\ s_{t+1} &= \Gamma_t(\overline{s}_t,z_{t+1})\end{aligned} \tag{5}$$

whereF_0^s is the distribution of initial states.

2.2 Neural networks

Our solution algorithm relies on approximating both the policy function for the continuous choices and the post-decision value function with neural networks. First, we introduce a value network for the post-decision value function with parameters $\theta_{\overline{v}}$, where the inputs are time t and the post-decision states $\overline{s}_t$, i.e.

$$\overline{v}_t\left(\overline{s}_t\right) \approx \overline{v}(t,\overline{s}_t;\theta_{\overline{v}}). \tag{6}$$

Second, we introduce a policy neural network with parameters θ_{π^a} for the continuous choices, where the inputs are time t and states s_t, and the outputs are the continuous choices a_t for each of the potential discrete choices d_t, i.e.

$$\left\{a_t^0, a_t^1, \ldots, a_t^{\#_d - 1}\right\} = \pi^a\left(t, s_t; \theta_{\pi^a}\right). \tag{7}$$

For simplicity we write $\pi^a\left(t, s_t, d_t; \theta_{\pi^a}\right)$ to pick the continuous choice associated with a specific discrete choice.

We do not need to parameterize the discrete policy function π^d directly. Defining the value-of-choice function

$$\mathcal{V}(t, s_t, d_t, a_t; \theta_{\overline{v}}) = u_t(s_t, d_t, a_t) + \beta \begin{cases} h(\overline{s}_t) & \text{if } t = T-1 \\ \overline{v}_t\left(t, \overline{s}_t; \theta_{\overline{v}}\right) & \text{else,} \end{cases} \tag{8}$$
$$\overline{s}_t = \overline{\Gamma}_t\left(s_t, d_t, a_t\right),$$

the optimal discrete choice can be calculated as

$$\pi_t^d(s_t, \varepsilon_t^d; \theta_{\overline{v}}, \theta_{\pi^a}) = \arg\max_{d_t^j \in \mathcal{D}_t(s_t)} \mathcal{V}(t, s_t, d_t^j, a_t^j; \theta_{\overline{v}}) + \varepsilon_t^d(d_t^j), \tag{9}$$
$$\left(a_t^0, a_t^1, \ldots, a_t^{\#_d - 1}\right) = \pi^a\left(t, s_t, \theta_{\pi^a}\right),$$

or using the Gumbel assumption choice probabilities can be derived as

$$\Pr[d_t = d^j \mid t, s_t] = \frac{\exp\left(\mathcal{V}(t, s_t, d_t^j, a_t^j; \theta_{\overline{v}})/\sigma_\varepsilon\right)}{\sum_{k=0}^{\#_d - 1} \exp\left(\mathcal{V}(t, s_t, d_t^k, a_t^k; \theta_{\overline{v}})/\sigma_\varepsilon\right)} \tag{10}$$

To compute the optimal discrete choice in (7), we need to compute the continuous choices conditional for each of the potential discrete choices. This explains our network structure, where the policy network outputs all the continuous choices for each discrete choice simultaneously. This works well in models with not too many discrete choices. In models with a high number of discrete choices, an alternative could be to directly approximate value-of-choice, i.e. $\mathcal{V}_t(s_t, d_t, a_t)$, though this has both the states and all discrete and continuous choices as inputs and therefore is more complex.

2.3 Training sample

For given value and policy parameters, $\theta_{\overline{v}}$ and θ_{π^a}, we can simulate forward starting from some distribution of initial states. We introduce exploration in the simulation to ensure that we visit more of the state space than the current parameters imply. We denote an exploratory version of the policy function for the continuous choices by $\breve{\pi}(t, s_t, d_t, \varepsilon_t^a; \theta_{\pi^a})$, where ε_t^a is a vector of exploration noise, drawn from the distribution $F_{\varepsilon^a}(s_t)$, and we ensure that there is no exploration at $\varepsilon_t^a = 0$ and the continuous choices are always feasible,

1. $\breve{\pi}^a(t, s_t, d_t, 0; \theta_{\pi^a}) = \pi^a(t, s_t, d_t, \theta_{\pi^a})$.
2. $a_t^j \in \mathcal{A}_t(s_t, d_t^j)$ for all $a_t^j \in \breve{\pi}^a(t, s_t, d_t, \varepsilon_t^a; \theta_{\pi^a})$.

We introduce exploration in the discrete choice by drawing larger taste shocks using a larger scale parameter $\breve{\sigma}_\varepsilon > \sigma_\epsilon$.

Our simulation procedure then is as follows

$$\begin{aligned} s_0 &\sim F_0^s \\ \varepsilon_t^d &\sim \text{i.i.d. Gumbel}(0, \breve{\sigma}_\varepsilon) \\ d_t &= \pi^d(t, s_t, \varepsilon_t^d; \theta_{\overline{v}}, \theta_{\pi^a}) \\ \varepsilon_t^a &\sim F_{\varepsilon^a}(s_t) \\ a_t &= \breve{\pi}^a(t, s_t, d_t, \varepsilon_t^a; \theta_{\pi^a}) \\ \overline{s}_t &= \overline{\Gamma}_t(s_t, d_t, a_t) \\ z_{t+1} &\sim F_t^z, \\ s_{t+1} &= \Gamma_t(\overline{s}_t, z_{t+1}) \end{aligned} \tag{11}$$

We denote the sample of initial states and shocks by

$$\underline{\mathcal{S}} = \left\{ s_{0,i}, z_{t,i}, \varepsilon_{t,i}^d, \varepsilon_{t,i}^a \, \forall i, \forall t \right\}, \tag{12}$$

and the full training sample by

$$\mathcal{S}\left(\theta_{\overline{v}}, \theta_{\pi^a}; \underline{\mathcal{S}}\right) = \left\{ s_{t,i}, \overline{s}_{t,i}, d_{t,i}, a_{t,i}, \varepsilon_{t,i}^d, \varepsilon_{t,i}^a \, \forall i, \forall t \right\}. \tag{13}$$

Total expected reward is then given by

$$R\left(\theta_{\overline{v}},\theta_{\pi^a},\underline{\mathcal{S}}\right)=\frac{1}{N}\sum_{i=0}^{N-1}\sum_{t=0}^{T-1}\left(\beta^t\left(u_t(s_{t,i},d_{t,i},a_{t,i})+\varepsilon^d_{t,i}(d_{t,i})\right)\right)+\beta^{T-1}h(\overline{s}_{T-1,i}). \tag{14}$$

2.4 DeepV

The general structure of our deep learning solution algorithm denoted DeepV, where the »V« refers to the Value function, is described in Algorithm 1. We start by drawing a random validation sample, $\underline{\mathcal{S}}^*$, with no exploration ($\breve{\sigma}_\varepsilon=\sigma_\varepsilon$, $\varepsilon^a_{t,i}=0$), and initial guesses of the network parameters, $\theta^{-1}_{\pi^a}$ and $\theta^{-1}_{\overline{v}}$.

The algorithm is then divided into episodes. In each episode, we first simulate a new training sample (step 1) and update our replay buffer based on this (step 2). The replay buffer is a set that contains the history of past simulated training samples, and is updated on a »first-in, first-out« basis.[4]

Secondly, we then train the neural network using a random training batch from the replay buffer (step 3). We also calculate an out-of-sample expected reward to keep track of progress (step 4) each Δ_R'th episode, and update the best the neural network parameters yet seen.

We repeat these steps for K episodes.[5] In our applications, we look at the time path of the out-of-sample expected reward, R^*, to ensure that no further improvement of the found policy can be achieved.

2.4.1 Updating network parameters

We now go into detail with the updating of network parameters in step 3 of Algorithm 1. To do so, we first need a target for the value network. Algorithm 2 describes how, given the post-decision state$\overline{s}_{t,i}$ from the training sample, and network parameters $\theta_{\overline{v}}$ and θ_{π^a}, can compute such a value target denoted $\tilde{\overline{v}}_{t,i}$. The expectation over

[4] The replay buffer implies that training sample is not fully based on the most recent guess on the policy network. This can be considered a form of exploration. See Druedahl and Røpke (2026) for a further discussion.

[5] The algorithm does not rely on backward induction. Backward induction requires T separate neural networks and also works poorly with simulation methods where multiple backward passes would be necessary. Therefore we solve the problem simultaneously across all periods which allows the network to learn the time-dependencies in the model directly.

Algorithm 1 DeepV solution algorithm

```
Draw sample of fixed initial states and shocks with no exploration ($\breve{\sigma}_\varepsilon = \sigma_\varepsilon$, $\varepsilon^a_{t,i} = 0$),
$\underline{\mathcal{S}}^*$ and initialize $R^* = -\infty$.
Draw random initial parameter values, $\theta^{-1}_{\overline{v}}$and $\theta^{-1}_{\pi^a}$ .
Initialize buffer memory $\mathcal{B} = \emptyset$.
For each episode $k = 0, 1, \ldots, K-1$ do:
    1. Simulate random training sample with exploration, $\mathcal{S}^k$.
    2. Update replay buffer $\mathcal{B}$ with $\mathcal{S}^k$ (»first in, first out«).
    3. Update network parameters to $\theta^k_{\pi^a}$ and$\theta^k_{\overline{v}}$ using random training batch $\mathcal{B}^k$.
    4. If $k$ is divisible by $\Delta_R$ or $k = K-1$ calculate out-of-sample expected reward,
       $R^k = R\left(\theta^k_{\overline{v}}, \theta^k_{\pi^a}, \underline{\mathcal{S}}^*\right)$.
       If $R^k > R^*$ set $\theta^*_{\pi^a} = \theta^k_{\pi^a}$ and $\theta^*_{\overline{v}} = \theta^k_{\overline{v}}$.
Return$\theta^*_{\pi^a}$ and$\theta^*_{\overline{v}}$ as the solution.
```

z_{t+1} is calculated with some form of numerical integration (quadrature or Monte Carlo). After step 1, we have a set of targets for training the value function,$\tilde{\overline{v}}_{t,i}$. Here we use the convention that variable with a tilde $\tilde{x}$ is computed within the algorithm and is not an input to the algorithm.

Algorithm 3 then describes the full updating of the parameters in the value and the policy networks. In this algorithm, we use the idea of target networks to stabilize the training process. In the first episode, we initialize the target networks with the same parameters as the initial guesses for the actual networks, $\check{\theta}^{-1}_{\overline{v}} = \theta^{-1}_{\overline{v}}$ and $\check{\theta}^{-1}_{\pi^a} = \theta^{-1}_{\pi^a}$. Later, we will be updating the target network more slowly using a weight of $\tau \in (0, 1]$ on the new parameters and $1 - \tau$ on the existing target network parameters.

In step 1, we compute the post-decision target value for all agents in our sample, across all periods, using the lagged value and policy target networks.

In step 2, the parameters in the value network $\theta_{\overline{v}}$ are updated by minimizing the mean squared error between the value network prediction and the targets $\tilde{\overline{v}}_{t,i}$ from step 2. We use a gradient descent type of algorithm and make $\#_{\overline{v}}$ updates using the full training sample. At the end of step 2 the target value parameters are updated.

In step 3, we update the policy parameters, simply using the value-of-choice as the loss function. We again use a gradient descent type of algorithm and make $\#_{\pi^a}$ updates using the full training sample. At the end of step 3 the target policy parameters are updated. We skip this step for the first $\underline{k}_{\pi^a} > 0$ iterations as the value function approximation initially might be too imprecise for policy optimization to be mean-

Algorithm 2 Value target

Given post-decision state$\overline{s}_{t,i}$ and network parameters $\theta_{\overline{v}}$ and θ_{π^a} compute

$$\tilde{\overline{v}}_{t,i} = \sigma_\varepsilon \log \left[\sum_{j \in \{0,1,\ldots,\#_d-1\}} \exp\left(\tilde{\overline{v}}^j_{t,i} / \sigma_\varepsilon \right) \right]$$

where

$$\tilde{\overline{v}}^j_{t,i} = \mathbb{E}^z_t \left[u_{t+1}\left(\tilde{s}_{t+1,i}, \tilde{d}^j_{t+1}, \tilde{a}^j_{t+1,i} \right) + \beta \begin{cases} h(\tilde{\overline{s}}_{t+1,i}) & \text{if } t = T-2 \\ \overline{v}\left(t+1, \tilde{\overline{s}}_{t+1,i}; \theta_{\overline{v}}\right) & \text{else,} \end{cases} \right]$$

for $j \in \{0, 1, \ldots, \#_d - 1\}$ where

$$\begin{aligned} \tilde{z}_{t+1,i} &\sim F^z_t\left(\overline{s}_{t,i}\right) \\ \tilde{s}_{t+1,i} &= \Gamma_t(\overline{s}_{t,i}, \tilde{z}_{t+1,i}) \\ \left(\tilde{a}^0_{t+1,i}, \tilde{a}^1_{t+1,i}, \ldots, \tilde{a}^{\#_d-1}_{t+1,i} \right) &= \pi^a(t+1, \tilde{s}_{t+1,i}, \tilde{d}_{t+1,i}; \theta_{\pi^a}) \\ \tilde{\overline{s}}_{t+1,i} &= \overline{\Gamma}_{t+1}(\tilde{s}_{t+1,i}, \tilde{a}^j_{t+1,i}, \tilde{d}^j_{t+1}) \end{aligned}$$

.

ingful.

We train the policy across all potential discrete choices, $d_t \in \mathcal{D}_t(s_t)$. If we were only training on the discrete choices from the simulation, our approximation of non-chosen values might continue to be bad, and a lot of exploration could be needed to avoid getting stuck in sub-optimal discrete choices.

2.4.2 Why the post-decision value function?

We could also have written the Bellman equation as

$$\underline{v}_t(s_t) = \sigma_{\varepsilon,t}(s_t) \log \left[\sum_{j \in \{1,2,\ldots,\#_d\}} \exp\left(\frac{v_t(s_t, d^j_t)}{\sigma_{\varepsilon,t}(s_t)} \right) \right] \tag{17}$$

$$v_t(s_t, d_t) = \max_{a_t \in \mathcal{A}_t(s_t, d_t)} u_t(s_t, d_t, a_t) + \beta \begin{cases} h(\overline{s}_t) & \text{if } t = T-1 \\ \mathbb{E}_t\left[\underline{v}_{t+1}(s_{t+1}) \right] & \text{else,} \end{cases} \tag{18}$$

where $\underline{v}_t(s_t)$ is the beginning-of-period value function. Instead of approximating $\overline{v}_t$, we could approximate v_t or $\underline{v}_t$. The downside of this would be that we in step 3

Algorithm 3 **Updating parameters**

```
If $k = 0$, set target network parameters $\check{\theta}_{\overline{v}}^{-1} = \theta_{\overline{v}}^{-1}$ and $\check{\theta}_{\pi^a}^{-1} = \theta_{\pi^a}^{-1}$.
  1. Compute the value targets $\tilde{\overline{v}}_{t,i}$ for all $\overline{s}_{t,i} \in \mathcal{B}^k$ with $t < T-1$ using Algorithm (2) with the lagged target network parameters, $\check{\theta}_{\pi^a}^{-1}$ and $\check{\theta}_{\overline{v}}^{-1}$.
  2. Update the value network to $\theta_{\overline{v}}^k$ for $\#_{\overline{v}}$ epochs using the loss function
       $L_{\overline{v}}\left(\theta_{\overline{v}}; \mathcal{B}^k\right) = \frac{1}{\left|\mathcal{B}^k\right|} \sum_{i,t,\overline{s}_{t,i} \in \mathcal{B}^k} \left(\overline{v}\left(t, \overline{s}_{t,i}; \theta_{\overline{v}}\right) - \tilde{\overline{v}}_{t,i}\right)^2$   (15)
     Update the value target parameters: $\check{\theta}_{\overline{v}}^k = \tau\theta_{\overline{v}}^k + (1-\tau)\check{\theta}_{\overline{v}}^{k-1}$.
  3. If $k > \underline{k}_{\pi^a}$ : Update the policy network to $\theta_{\pi^a}^k$ for $\#_\pi$ epochs using the loss function
       $L_{\pi^a}(\theta_{\pi^a}; \mathcal{B}^k) = -\frac{1}{\left|\mathcal{B}^k\right| \#_d} \sum_{i,t,s_{t,i} \in \mathcal{B}^k} \sum_{d_{t,i}^j \in \mathcal{D}_t(s_{t,i})} v_{t,i,j}$   (16)
     where
       $v_{t,i,j} = u_t(d_{t,i}^j, \tilde{a}_{t,i}^j, d^j) + \beta \begin{cases} h(\tilde{\overline{s}}_{t,i}) & \text{if } t = T-1 \\ \overline{v}\left(t, \tilde{\overline{s}}_{t,i}; \theta_{\overline{v}}^k\right) & \text{else,} \end{cases}$
       $\left(\tilde{a}_{t,i}^0, \tilde{a}_{t,i}^1, \ldots, \tilde{a}_{t,i}^{\#_d - 1}\right) = \pi(t, s_{t,i}; \theta_{\pi^a}^k)$
       $\tilde{\overline{s}}_{t,i} = \overline{\Gamma}_t(s_{t,i}, \tilde{a}_{t,i}, d_{t,i}^j).$
     Update the target policy parameters: $\check{\theta}_{\pi^a}^k = \tau\theta_{\pi^a}^k + (1-\tau)\check{\theta}_{\pi^a}^{k-1}$.
```

of Algorithm 3, we would need to keep track of derivatives inside the expectation operator when updating of θ_{π^a}. The upside is that the computation of the target in step 1 becomes easier, but this is not the computational bottleneck anyway.

2.4.3 Implementation

Implementing the solution algorithm in practice requires a number of choices:

1. Initialization and structure of the policy and value neural networks.
2. Final activation functions for policy network.
3. Size of training sample, replay buffer and training batch.
4. Optimizer, learning rates and number of epochs for the policy and value parameters.
5. Degree of exploration in simulation.
6. Degree of smoothing in target policy and value networks.
7. Termination criteria.

We explain this in detail in Appendix A.1. However, we note two things here. First, we structure time input as T dummies, i.e.

$$\boldsymbol{t}_{\text{dum}} = \{\mathbf{1}(t=0), \mathbf{1}(t=1), \ldots, \mathbf{1}(t=T-1)\}.$$

Second, we implement a version of ensemble learning by training multiple value networks at once. We draw independent initial parameters for $\#_{\overline{v}}$ different value networks indexed by m. In Algorithm 3, we update the value networks independent in step 2, and in step 3, we use the average across value networks, i.e.

$$\overline{v}\left(t, \tilde{\overline{s}}_{t,i}; \theta_{\overline{v}}^{k}\right) = \frac{1}{\#_{\overline{v}}} \sum_{m=0}^{\#_{\overline{v}}-1} \overline{v}\left(t, \tilde{\overline{s}}_{t,i}; \theta_{\overline{v}}^{m,k}\right). \tag{19}$$

All those value networks are receiving the same data batches, so that the only randomness averaged-out comes from the different initialization of each network. This turns out to improve performance.

2.4.4 Using information from first order conditions

We can extend our solution algorithm to include information from the first order conditions. We extend the value network to be approximating both the level of the post-decision value function and selected derivatives. In step 1, of Algorithm 3, we compute both the level of the post-decision value and the derivatives we need. In step 2, we update the new extended value network as before. In step 3, we train on both the value-of-choice and squared errors in the first order condition. The first order conditions should be zero even when they are not sufficient. The details are in Appendix A.2.

2.4.5 Final backward induction step

Once we have solved the model, we can verify the accuracy of the solution by testing, whether a single backward induction loop can improve the policy. First, we simulate a large training sample using the so-far found policy and value networks. We then step backward period by period training period-specific new policy and value networks correcting the previously found across-age networks. The details are in Appendix A.4.

2.4.6 Convex models

The DeepV algorithm can be simplified considerably for convex models without discrete choices. The policy network only needs to output a single vector of continuous choices, and taste shocks are irrelevant. Otherwise, the algorithm is the same.

3 Test Model: Multiple Durable Goods and Non-Convex Adjustment Costs

In this section, we consider a model which we can solve with both standard dynamic programming methods and with our deep learning algorithm. First, this allows us to verify that the deep learning algorithm can deliver an accurate solution. Secondly, we show that our deep learning algorithm suffers much less from the curse of dimensionality than a standard value function iteration by scaling up the dimensionality of the model.[6]

[6] We can easily test the convex version of the DeepV in the canonical Buffer-stock model. In Appendix A.3, we compare our deep learning solution with a very accurate Endogenous Grid Point (EGM) method, and the DeepFOC deep learning algorithm from Druedahl and Røpke (2026), which cannot

3.1 Model

We consider a model with a non-durable consumption good c_t and D durable consumption goods, $d_t = \{d_{1,t}, d_{2,t}, \ldots, d_{D,t}\}$ subject to non-convex adjustment costs. The household has CRRA preferences over a Cobb-Douglas aggregate of non-durable and durable consumption.

$$u(c_t, d_t) = \frac{\left(c_t^{1-\sum_{j=1}^{D}\omega_j} \prod_{j=1}^{D}\left(d_{j,t} + \underline{d}_j\right)^{\omega_j}\right)^{1-\rho}}{1-\rho} \tag{20}$$
$$\underline{d}_j > 0, \omega_j \in (0,1), \sum_{j=1}^{D}\omega_j < 1, \rho > 0, \rho \neq 1,$$

where, for numerical stability, there is a floor under durable consumption, $\underline{d}_j > 0$. The state variables are cash-on-hand m_t, permanent income p_t, and the beginning-of-period stock of each durable good $n_{j,t}$. In each period, the agent decides how much to consume c_t and how much of each durable good, $d_{j,t}$, to accumulate.

When adjusting a durable good ($d_{j,t} \neq n_{j,t}$), the household must sell the current durable stock with a proportional loss of $\nu_j \in (0,1)$. Consequently, whether or not to adjust can be considered a discrete choice. End-of-period assets are given by

$$\bar{b}_t = m_t + \sum_{j=1}^{D}\left[\left(1 - \nu_j \mathbf{1}(d_{j,t} \neq n_{j,t})\right) n_{j,t} - d_{j,t}\right] - c_t. \tag{21}$$

Borrowing is not allowed, $\bar{b}_t \geq 0$. The depreciation rates for the durable stocks are δ_j such that

$$n_{t+1,j} = (1-\delta_j) d_{j,t}. \tag{22}$$

The state-transition for p_t is

$$p_{t+1} = p_t^{\rho}\xi_{t+1}, \quad \log \xi_{t+1} \sim \mathcal{N}(-\sigma_{\xi}^2/2, \sigma_{\xi}^2), \tag{23}$$

where $\rho \in (0,1)$ is the AR(1) parameter and ξ_t is a permanent income shock. Income

be used for non-convex models. The baseline DeepV solution method is only slightly inferior to the EGM and DeepFOC methods in accuracy, and full equal accuracy is obtained when using multiple value networks and information from the first order conditions.

is then given by

$$y_{t+1} = p_{t+1}\psi_{t+1}, \quad \log \psi_{t+1} \sim \mathcal{N}(-\sigma_\psi^2/2, \sigma_\psi^2), \tag{24}$$

where ψ_{t+1} is a transitory income shock. Future cash-on-hand then is

$$m_{t+1} = (1+r)\overline{b}_t + y_{t+1}, \tag{25}$$

where $r > 0$ is the interest rate. The calibration is shown in Appendix B.1.

3.2 Implementation

We now turn to how we implement our deep learning algorithm in practice. We focus on the complicated case with $D = 2$ durable goods. Here there are four discrete options, $\iota_t \in \{1, 2, 3, 4\}$, specifically (i) keep both durables, (ii) adjust durable $d_{1,t}$, (iii) adjust durable $d_{2,t}$, and (iv) adjust both durables.

Policy network outputs. We set up the policy networks such that it has 8 outputs, and we use a *sigmoid* for the final activation function to ensure each output is between zero and one. The continuous policy outputs are linked to the discrete choices as follows:

1. No adjustment, $\iota_t = 1$: $\lambda_t^{e,1} \in [0, 1]$.
2. Adjust durable 1, $\iota_t = 2$: $\lambda_t^{e,2}, \lambda_t^{c,2} \in [0, 1]$.
3. Adjust durable 2, $\iota_t = 3$: $\lambda_t^{e,3}, \lambda_t^{c,3} \in [0, 1]$.
4. Adjust both durable 1 and 2, $\iota_t = 4$: $\lambda_t^{e,4}, \lambda_t^{c,4}, \lambda_t^{d,4} \in [0, 1]$.

To calculate the actual continuous choices based on the continuous policy outputs, we first calculate total resources available for spending,

$$x_t^{\iota_t} = m_t + (1-\nu_1)\mathbf{1}(\iota_t \in \{2, 4\}) + (1-\nu_2)\mathbf{1}(\iota_t \in \{3, 4\}) \tag{26}$$

and then use following formulas

$$c_t = \begin{cases} \lambda_t^{e,\iota_t} x_t^{\iota_t} & \text{if } \iota_t = 1 \\ \lambda_t^{e,\iota_t} \lambda_t^{c,\iota_t} x_t^{\iota_t} & \text{else} \end{cases} \tag{27}$$

$$d_{1,t} = \begin{cases} n_{1,t} & \text{if } \iota_t \in \{1,3\} \\ \lambda_{k,t}^{\iota_t} x_t^{\iota_t} & \text{else} \end{cases} \tag{28}$$

$$d_{2,t} = \begin{cases} n_{2,t} & \text{if } \iota_t \in \{1,2\} \\ \lambda_{k,t}^{\iota_t} x_t^{\iota_t} & \text{if } \iota_t = 3 \\ \left(1 - \lambda_{k,t}^{\iota_t}\right) x_t^{\iota_t} & \text{else} \end{cases} \tag{29}$$

$$\overline{b}_t = \left(1 - \lambda_t^{e,\iota_t}\right) x_t \tag{30}$$

This also automatically ensures that the borrowing constraint is fulfilled, $\overline{b}_t \geq 0$. This implementation is not unique. In our replication package, we consider an alternative, where the output of the neural network are budget shares. The performance is similar.

Network architecture. The value and policy networks both have two hidden layers with 500 neurons each. For activation functions, we use ReLU in intermediate layers for both networks. We train 3 independent value networks and average their predictions to reduce the bias from initialization. Learning rates are set to 10^{-3} for both networks and decay by a factor of 0.9999 per iteration down to a minimum of 10^{-5}. Training of the policy network begins after iteration 50 to allow the value networks to stabilize. The smoothing parameter for the target networks is $\tau = 0.20$.

Numerical integration. We use Gauss-Hermite quadrature with $N_\xi = 4$ nodes for permanent income shocks and $N_\psi = 4$ nodes for transitory shocks, implying $4 \times 4 = 16$ quadrature points per period.

Training and validation sample. Each iteration simulates a training sample of $N = 150$ households, and the batch size is the same. The replay buffer has a memory of $8 \cdot 150 = 1,200$. The average lifetime reward is evaluated every $\Delta_R = 10$ iterations using a validation sample of $N = 100,000$ agents.

Exploration. For continuous actions, the exploration noise in training is Gaussian with a fixed standard deviation of 0.10. We induce exploration across discrete choices by scaling the taste shocks with a factor of 1.05 in training.

The final details on the implementation are found in Appendix B.2.

3.3 Results

We run our deep learning algorithm for 120 minutes for $D = 1$ and 180 minutes for $D = 2$. We use a NVIDIA H100 GPU with 80GB RAM. We compare our deep learning solution to a standard dynamic programming solution using a parallelized version of VFI in C++, on a machine with 72 CPU cores (two Intel Xeon Gold 6254, 3.1 GHz). We use multi-starts of the Method of Moving Averages (MMA) to find the optimal choices for adjuster and keepers. We set the following grid sizes: $\#_a = 200$, $\#_m = \#_n = \#_x = 150$ and $\#_p = 100$. We then simulate the obtained solution on the same validation sample as in the deep learning algorithm. Appendix Table B.1 reports all the relevant hyperparameters used.

Average lifetime rewards.

We first evaluate the relative accuracy of the two solutions by comparing average lifetime rewards, i.e. average expected discounted utility. Specifically, we report a transfer-equivalent measure. This measure expresses the welfare difference between the deep-learning (DL) and dynamic programming (DP) solutions as the transfer in initial cash-on-hand that would make an agent following the DP policy indifferent to switching to the DL policy.

Figure 1 reports the required transfer in basis points of initial cash-on-hand over computational time. A positive value indicates that households would be willing to give up resources to use the DL solution rather than the DP benchmark. After roughly 120 minutes, the algorithm plateaus at an implied willingness to pay of about 4 basis points of initial cash-on-hand to use the DL solution rather than DP, in the 1D case, and 5 basis points in the 2D case. The vertical blue and orange lines mark the DP run times in the one – and two-dimensional cases, respectively. The DL method attains a similar accuracy to the DP solution after roughly 20 minutes for the 1D case and after 90 minutes for the 2D case. Both obtain a slightly higher objective at the end of training.

This comparison also illustrates how our algorithm alleviates the curse of dimen-

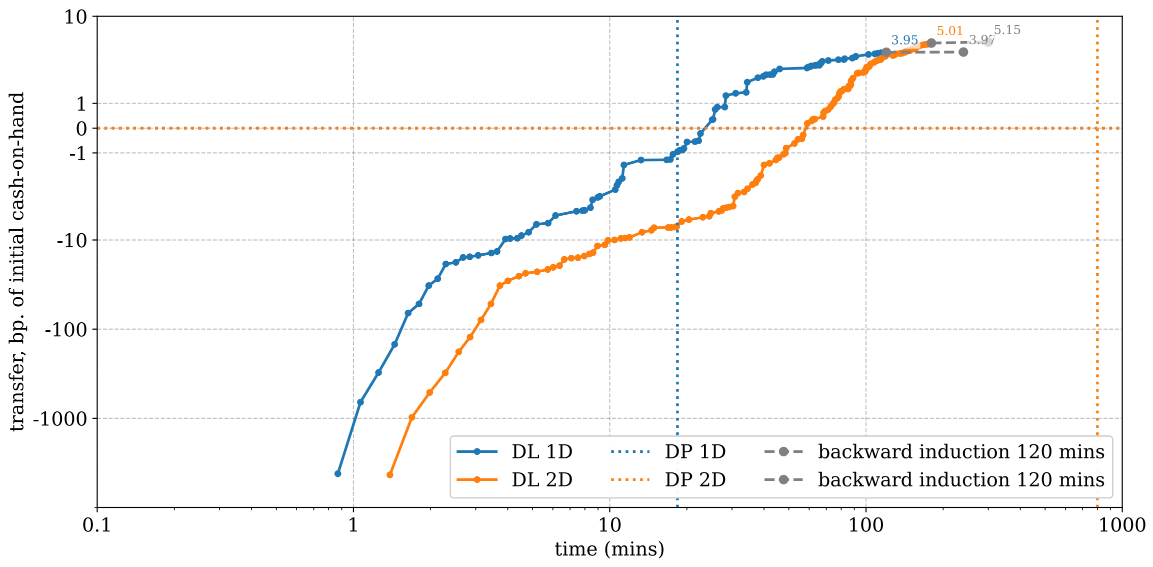

Figure 1: Non-Convex Durables Model: Convergence

Notes: The x-axis shows time in minutes, while the y-axis shows the transfer required to make agents indifferent between the DP solution and the current DL solution. When below 0, this means that the agent would have demanded a transfer of x units of cash-on-hand to use the DL solution instead of the DP one. When above 0, it means that the agent would have demanded a transfer of x units of cash-on-hand to use the DP solution instead of the DL one. We only show the transfer when the algorithm improves.

sionality. In DP, adding one durable good increases solving time from under 20 minutes to over 13 hours. By contrast, our DL algorithm requires only an additional 40 minutes to match DP accuracy. Section 3.5 shows that the extended algorithm targeting the first-order condition improves performance further, making the DL algorithm competitive with DP even in the single-durable-good case.

We also compute a single backward step starting from our baseline solution (see the algorithmic details in Appendix A.4) . This allows us to check whether it is possible to improve on our current solutions. Running a backward step for 120 minutes does not meaningfully improve our solution.

Euler-errors.

Figure 2 also confirms the accuracy of the deep learning method, plotting the log-10

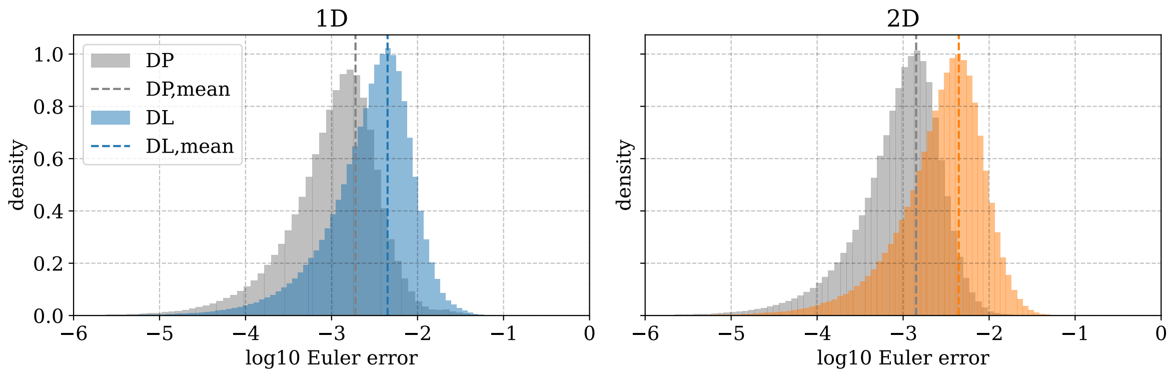

Figure 2: Non-Convex Durables Model: log-10 Euler errors

Notes: Each figure compares log-10 absolute Euler errors for the DL solution (orange) with the DP-solution (grey). We exclude constrained households from the sample if their post-decision cash-on-hand is sufficiently close to the borrowing constraint: $m_t \leq e^{-2}$, and households in the last period. Vertical dashed lines show the average errors in the sample.

absolute Euler errors in the validation sample for both methods, defined as

$$\log \text{Euler errors} = \log_{10} \left| \frac{u'^{-1}(\beta(1+r)\mathbb{E}[u'(c_{t+1,i}, d_{t+1,i})]}{c_{t,i}} - 1 \right|.$$

The average Euler error is slightly higher with our deep learning algorithm compared to the dynamic programming one, but remains below -2, so that, on average, households in our simulation sample make mistakes equivalent to less than 1 percent unit of consumption.

Life-cycle profiles.

We finally ensure that the life-cycle profiles of the two solutions are similar in terms of average, percentiles and distributions of state and choice variables. Figures 3 and 4 plot the average, 25th and 75th percentiles life-cycle profiles in the model, for both solutions. We find no visible differences between the two, a fact confirmed by Tables 1 and 2, which computes the key moments and correlations for each state and choice variables for the model with 2 durable goods.[7]

Similar average outcomes may however mask distributional differences. We thus also plot the differences in the cross-section of cash-on-hand over time for both so-

[7] For the 1D case, see Appendix Tables B.2 and B.3.

	t =	0		4		9		14		19	
	Model	DP	DL	DP	DL	DP	DL	DP	DL	DP	DL
var	moment										
m	Mean	1.00	1.00	1.09	1.09	1.17	1.16	1.11	1.11	1.00	1.00
	Median	0.99	0.99	1.05	1.05	1.10	1.10	1.05	1.04	0.95	0.95
	Variance	0.01	0.01	0.08	0.08	0.15	0.15	0.15	0.15	0.11	0.11
	Skewness	0.30	0.30	0.86	0.86	1.16	1.14	1.21	1.22	1.02	1.04
	Kurtosis	3.16	3.16	4.34	4.35	5.47	5.28	5.60	5.67	4.89	4.97
c	Mean	0.58	0.59	0.77	0.77	0.82	0.82	0.89	0.89	1.18	1.18
	Median	0.58	0.59	0.76	0.76	0.80	0.80	0.86	0.86	1.12	1.13
	Variance	0.00	0.00	0.02	0.02	0.04	0.04	0.06	0.06	0.14	0.13
	Skewness	0.64	0.46	0.54	0.55	0.68	0.69	0.82	0.83	0.96	1.00
	Kurtosis	3.86	3.87	3.51	3.54	3.81	3.76	4.17	4.24	4.53	4.80
d_1	Mean	0.28	0.27	0.63	0.63	0.76	0.76	0.73	0.73	0.28	0.28
	Median	0.28	0.27	0.61	0.61	0.74	0.74	0.70	0.70	0.22	0.23
	Variance	0.00	0.00	0.04	0.04	0.06	0.06	0.06	0.06	0.02	0.02
	Skewness	0.24	0.63	0.52	0.49	0.59	0.60	0.88	0.82	1.38	1.37
	Kurtosis	4.33	4.13	3.28	3.23	3.35	3.59	4.26	3.91	5.07	5.17
d_2	Mean	0.13	0.14	0.34	0.34	0.41	0.40	0.40	0.40	0.16	0.15
	Median	0.13	0.14	0.34	0.34	0.39	0.39	0.38	0.38	0.13	0.12
	Variance	0.00	0.00	0.01	0.01	0.02	0.02	0.02	0.02	0.01	0.01
	Skewness	-1.12	-1.74	0.26	0.21	0.56	0.62	0.84	0.94	1.35	1.33
	Kurtosis	6.27	7.76	3.07	3.12	3.46	3.79	4.20	4.88	5.30	5.17

Table 1: Moments of the Non-Convex Durables Model (2-durables)

lutions, in Figure 5 (Appendix Figure B.2 for the 1D case). If both solutions were exactly the same, all the points would perfectly align on the 45° line. We find a few differences, but the correlation is very high (above 0.99 in all periods), and the mean-absolute relative difference low.

Similarly, Figure 6 plots the share of households who do the same discrete choice in both solutions. This share is very close to 1 and relatively constant over time, again indicating that our solution seems robust.

3.4 The role of exploration noise in simulation for convergence

The previous section mentioned the key role of exploration noise for visiting parts of the state space outside the optimal path. Figure 7 confirms this: Without exploration noise convergence (we set exploration noise for continuous actions to zero,

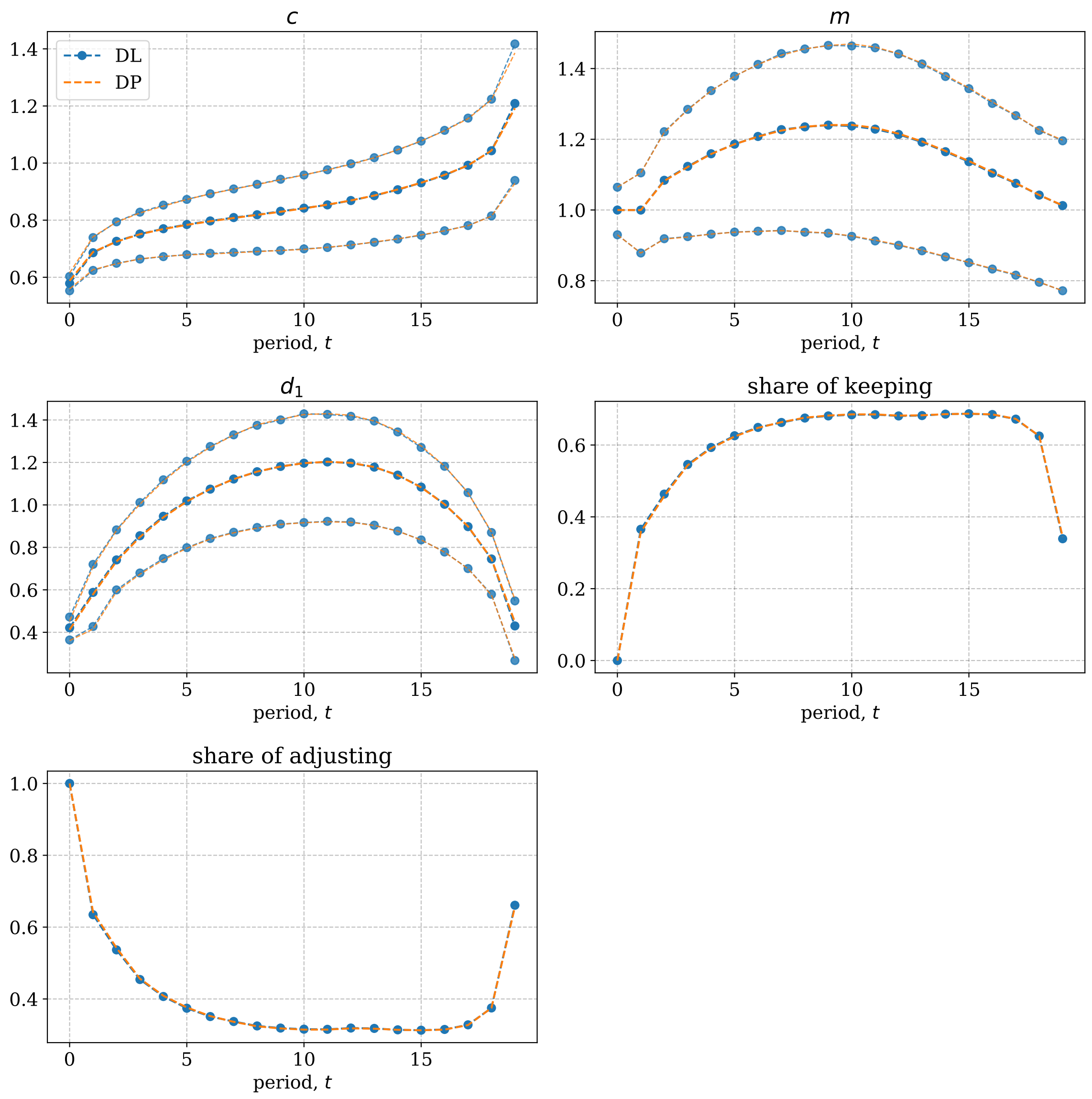

Figure 3: Non-Convex Durables Model: Life cycles profiles

Notes: Each figure compares the DL solution (full line) with the DP-solution (dashed line). For consumption, cash-on-hand and durables, the plot shows both the average life cycle profile (in a darker color, in the center), and the bottom 25% and top 75% percentiles, in a lighter shade.

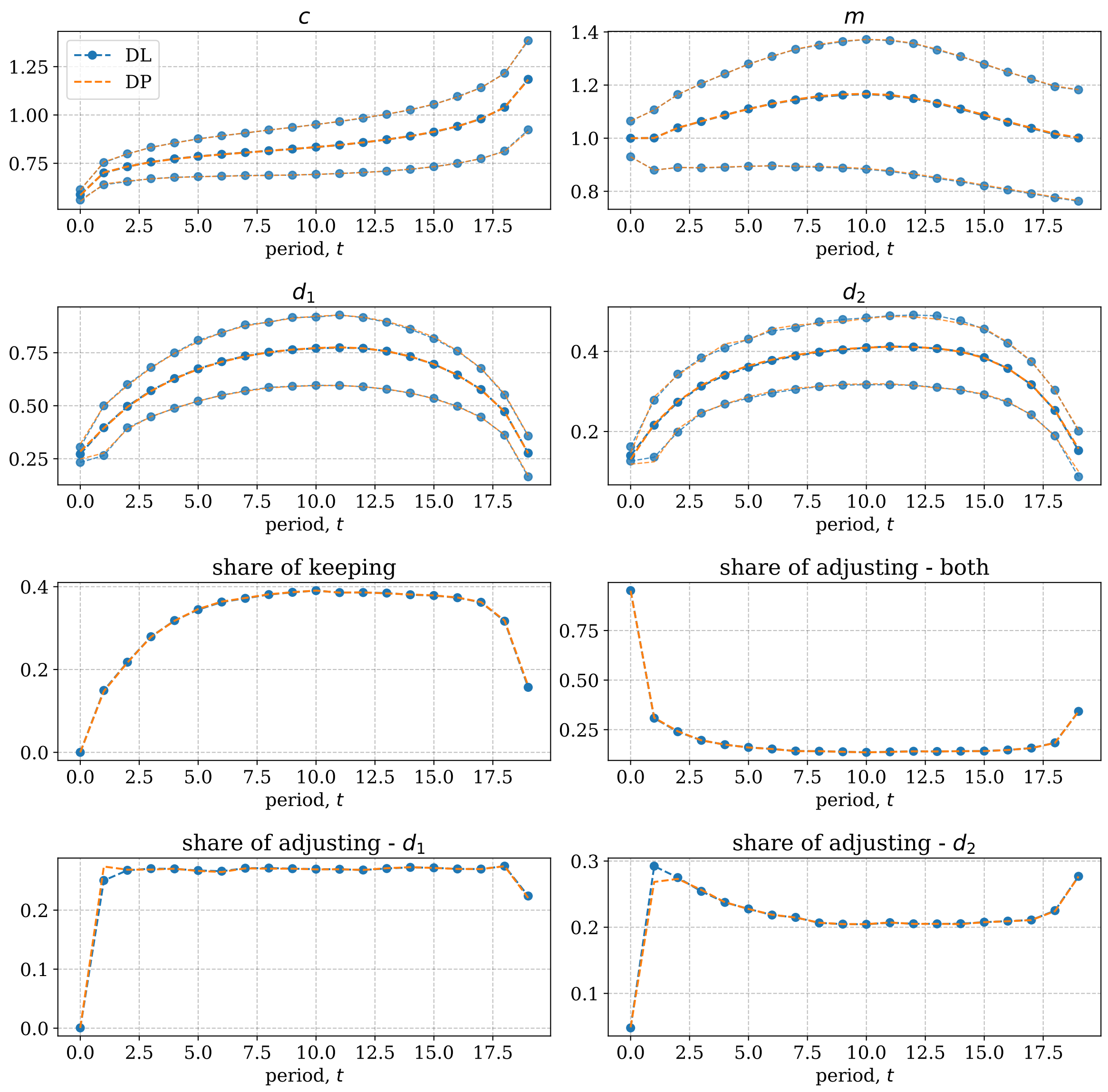

Figure 4: Non-Convex Durables Model: Life cycles profiles

Notes: Each figure compares the DL solution (blue dashed-line with circles) with the DP-solution (orange dashed line). The middle line shows the average value in the sample, while the bottom (top) line shows the bottom 25th (top 75th) percentile values.

t =	0		4		9		14		19	
Model	DP	DL	DP	DL	DP	DL	DP	DL	DP	DL
Corr(m,c)	0.67	0.67	0.84	0.84	0.88	0.88	0.92	0.92	0.93	0.93
Corr(m,d_1)	0.90	0.88	0.48	0.48	0.60	0.59	0.73	0.73	0.56	0.58
Corr(m,d_2)	0.59	0.59	0.46	0.46	0.65	0.64	0.77	0.76	0.53	0.53
Corr(m,p)	0.01	0.01	0.80	0.80	0.82	0.83	0.87	0.87	0.94	0.94
Corr(c,d_1)	0.51	0.46	0.45	0.45	0.67	0.67	0.80	0.79	0.37	0.39
Corr(c,d_2)	-0.06	0.01	0.49	0.49	0.72	0.71	0.82	0.81	0.45	0.45
Corr(c,p)	0.62	0.65	0.97	0.96	0.98	0.98	0.98	0.97	0.90	0.90
Corr(d_1,d_2)	0.40	0.33	0.16	0.15	0.52	0.52	0.71	0.70	0.25	0.26
Corr(d_1,p)	-0.27	-0.33	0.52	0.52	0.69	0.68	0.79	0.78	0.55	0.57
Corr(d_2,p)	-0.29	-0.19	0.50	0.49	0.71	0.70	0.78	0.78	0.53	0.53

Table 2: Correlations of the Non-Convex Durables Model (2-durables)

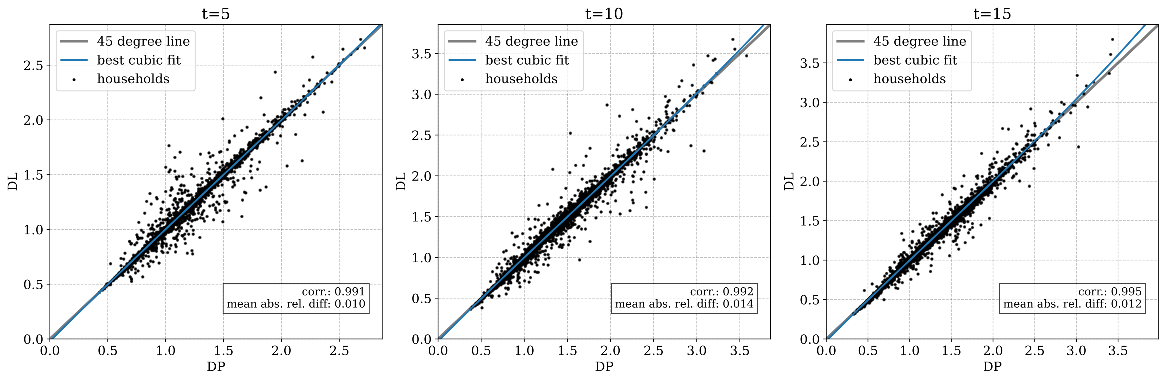

Figure 5: Cross-Section of cash-on-hand in the DL and DP Solutions for $D = 2$

Notes: Each panel shows the distribution of cash-on-hand in the DP vs DL solution, for a different period. If both solutions perfectly coincided, all the points would lie on the 45° line.

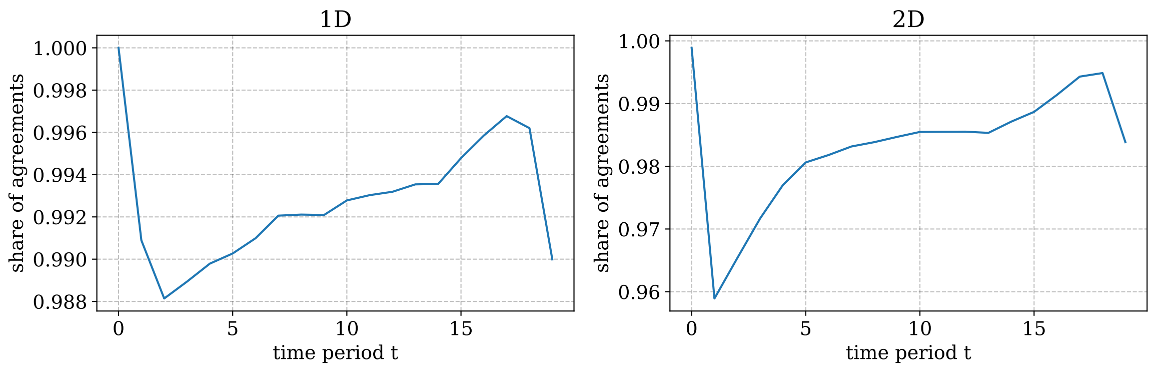

Figure 6: Share of cases where DP and DL yields same discrete choice across time

Notes: Each panel shows the share of agents making the same discrete choice across both DP and DL solution methods, across periods.

and don't increase the variance of taste shocks during simulation) is much slower and a solution as good as the DP benchmark is never reached. The intuition relates to the moving-target nature of our algorithm. The policy network is updated using the value network, which is itself trained on data generated by the current policy. For the policy update to be effective, the post-decision value function must be accurately approximated not only at the states visited under the current policy, but also in a neighborhood around them — otherwise, gradient-based policy improvements have no reliable signal to follow. Without exploration, the value network is only trained on states along the current (sub-optimal) policy path, creating a circular dependency: inaccurate off-path value estimates prevent the policy from improving, and the unchanged policy never generates off-path data. Exploration noise breaks this circularity by enriching the training sample with states beyond those implied by the current policy, allowing the value network to generalize over a broader region of the state space and thereby accelerating convergence. We also find that adding more noise (twice the size of exploration noise for continuous actions and for taste-shocks during simulation) to our baseline algorithm slightly improves the solution, but by a relatively small amount, suggesting that our current level of exploration noise is appropriate.

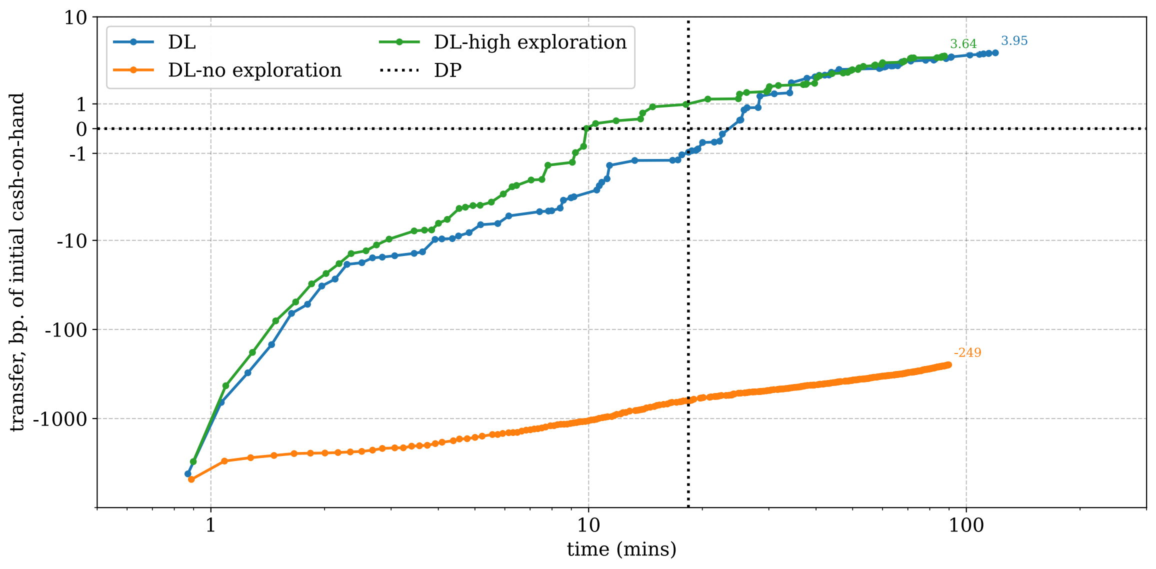

Figure 7: Non-Convex Durables Model: Convergence

Notes: The x-axis shows time in minutes, while the y-axis shows the transfer required to make agents indifferent between the DP solution and the current DL solution for three algorithms: our benchmark case, in blue, the same algorithm without any exploration noise, in orange, and a higher value of exploration noise in green. We only show the transfer when the algorithm improves.

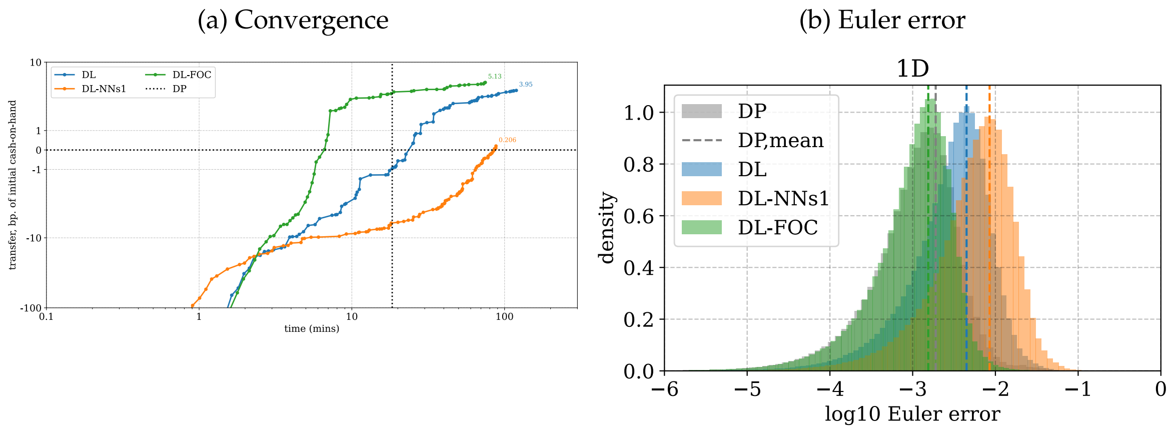

Figure 8: Using information from first order conditions ($D = 1$)

Notes: These figures show a comparison of a DP solution, our baseline DL solution and a DL extended with information from the Euler-equation. See Figure 1 and 2 for details on each plot.

3.5 Information from first-order conditions and ensemble learning

We now evaluate the two extensions introduced in Section 2.4.4: Targeting the Euler equation and training multiple neural networks to reduce initialization bias (algorithmic details in Appendix A.2). Figure 8 compares three specifications: (i) our baseline with ensemble learning but without FOC targeting, (ii) a model adding the FOC as a target, and (iii) a model with neither FOC targeting nor multiple networks. The extended algorithm converges faster and yields smaller Euler equation errors than both the alternative DL and DP solutions.

4 A Large-Scale Non-Convex Life-Cycle Model

We now consider a large-scale life-cycle model, where households make decisions regarding their labor supply, consumption, housing, and financial portfolio of risky and safe assets. This model cannot be solved with conventional methods, but can still be solved with our deep learning algorithm in a feasible number of hours on a single GPU.

4.1 Model

The model is divided into three blocks, (i) employment, (ii) housing and mortgages, and (iii) portfolio choices. Each of these blocks entails a discrete choice, respectively denoted d_t^e, d_t^h and d_t^p, or together $d_t = (d_t^e, d_t^h, d_t^p)$. First, the states, s_t, for the household problem are

1. Employment block:
   (a) Job-opportunity status, e_t,
   (b) Human capital, p_t,
   (c) Current wage, w_t,
2. Housing block:
   (a) Current house price, p_t^h,
   (b) Housing stock, h_t,
   (c) Mortgage balance, L_t,
3. Portfolio block:
   (a) Risky financial assets, a_t,
   (b) Risk-free financial assets, b_t.

Secondly, depending on its discrete choice, the household make a subset of the following continuous choices:

1. Consumption, c_t,
2. Housing stock, $\bar{h}_{t+1}$,
3. Mortgage balance, $\bar{L}_{t+1}$,
4. Risky financial assets, $\bar{a}_t$,
5. Risk-free financial assets, $\overline{b}_t$,
6. Housing services, $\tilde{h}_t$, $(= \bar{h}_t$ for owners and buyers).

Formally, we denote the discrete choice set as $\mathcal{D}(s_t)$ and the implied continuous choice set as $\mathcal{Y}(s_t, d_t)$ such that:

$$d_t \in \mathcal{D}(s_t), y_t \in \mathcal{Y}(s_t, d_t).$$

States and choices imply the post-decision states, $\bar{s}_t = \overline{\Gamma}(s_t, d_t, y_t)$, which are:

1. Employment status, $\bar{\ell}_t$,
2. Housing stock, $\bar{h}_t$,
3. Mortgage balance, $\bar{L}_t$,
4. Risky financial assets, $\bar{a}_t$ (end-of-period),
5. Risk-free financial assets, $\bar{b}_t$ (end-of-period).

The stochastic shocks, z_t, are:

1. Human capital shock, ε_t^p,
2. Job-offer shock, ε_t^e,
3. Wage offer, ω_t,
4. Job-destruction shock, ε_t^u,
5. Risky rate, r_t^a.

Together, the post-decision states and the shocks determine next period states, $s_{t+1} = \Gamma(\bar{s}_t, z_{t+1})$. Preferences are a Cobb-Douglas aggregate of non-housing consumption c_t and housing services $\tilde{h}_t$, with a disutility of work χ_e. Per-period utility is

$$u(c_t, \tilde{h}_t, \ell_t) = \frac{(c_t^\alpha \tilde{h}_t^{1-\alpha})^{1-\sigma}}{1-\sigma} - \chi_e \ell_t, \tag{31}$$

where $0 < \alpha < 1$ and $\sigma > 0$. The household maximizes expected lifetime utility

$$\max_{d_t \in \mathcal{D}(s_t), y_t \in \mathcal{Y}(s_t, d_t)} \mathbb{E}_0 \left[\sum_{t=0}^{T-1} \beta^t u(c_t, \tilde{h}_t, \ell_t) \right]. \tag{32}$$

The recursive formulation is

$$v_t(s_t) = \max_{d_t \in \mathcal{D}(s_t), y_t \in \mathcal{Y}(s_t, d_t)} u(c_t, \tilde{h}_t, \ell_t) + \beta \mathbb{E}_t[v_{t+1}(s_{t+1})], \tag{33}$$

subject to the state and post-state transition functions, and with terminal value $v_T(\cdot) = 0$.

Budget constraint The total available resources are

$$m_t = m_t^e + m_t^h + m_t^p, \tag{34}$$

where m_t^e, m_t^h, and m_t^p denote the net cash flows from the employment, housing, and portfolio blocks. Let x_t^h denote the funds allocated to housing expenses, and x_t^p the funds allocated to financial saving. The per period budget constraint is

$$c_t + x_t^h + x_t^p = m_t. \tag{35}$$

i) Employment block. The labor market follows a job-ladder structure in which the household chooses whether to work ($d_t^e = E$) or not ($d_t^e = U$). If employed, earnings are labor income equal to the product of its human capitalp_t and current wage w_t. If not employed, it receives unemployment benefits $\min\{\chi^u p_t, \bar{u}\}$, where $\bar{u}$ is a cap on unemployment benefits. The cash-on-hand from the job ladder block is therefore

$$m_t^e = \begin{cases} \min\{\chi^u p_t, \bar{u}\} & \text{if } d_t^e = U \\ p_t w_t & \text{if } d_t^e = E \end{cases}. \tag{36}$$

Human capital evolves stochastically, with drift parameters that differ by employment status:

$$\log p_{t+1} = \log p_t + \sigma^p \varepsilon_t^p + \begin{cases} \mu^{p,u} & \text{if } d_t^e = U \\ \mu^{p,e} & \text{if } d_t^e = E \end{cases}, \tag{37}$$

where $\mu^{p,e} \geq \mu^{p,u}$, and ε_t^p is i.i.d. normal.

Each period, households may decide to work ($d_t^e = E$) only if they have a job opportunity available to them $e_t = 1$,

$$d_t^e \in \begin{cases} \{E, U\} & \text{if } e_t = 1 \\ \{U\} & \text{if } e_t = 0 \end{cases}. \tag{38}$$

Employment status is

$$\ell_t = \mathbf{1}\{d_t^e \in \{E\}\}. \tag{39}$$

Households are subject to a job-offer shock $\varepsilon_t^e = \{0, 1\}$ with probabilities $(1 - \pi^e)$, π^e

and a job-destruction shock $\varepsilon_t^u = \{0,1\}$ with probabilities $(1-\delta^e), \delta^e$, such that

$$\Pr\left[e_{t+1} = 1 \,|\, d_t^e\right] = \begin{cases} 1-\delta^e & \text{if } d_t^e = E \\ \pi^e & \text{if } d_t^e = U \end{cases} \tag{40}$$
$$\Pr\left[e_{t+1} = 0 \,|\, d_t^e\right] = \begin{cases} \delta^e & \text{if } d_t^e = E \\ 1-\pi^e & \text{if } d_t^e = U. \end{cases}$$

Each job-offer has a wage ω_t drawn from F_ω. Households who are working this period and receive a new job-offer automatically accept a new job offer if $\omega_t > w_t$. In sum,

$$w_t = \begin{cases} \omega_t & \text{if } e_t = 0 \\ \max\{\omega_t, w_t\} & \text{if } e_t = 1 \end{cases}. \tag{41}$$

ii) Housing block. Housing decisions are made over both tenure and finance. At the start of the period the household chooses whether to rent ($d_t^h = R$), remain an owner without adjusting ($d_t^h = O$), buy a new house ($d_t^h = B$). Renters choose how much to rent, $\tilde{h}_t$ and buyers choose how much to buy, h_{t+1}. Buyers also choose mortgage size L_{t+1}. The net cash-flow effect of the housing block is

$$m_t^h = \begin{cases} \max(0, (p^h(1-\tau^h) - \delta^h)h_t - L_t(1+r^L)) & \text{if } d_t^h = R, \\ -\delta^h h_t - f_t(L_t) & \text{if } d_t^h = O, \\ (p_t^h(1-\tau^h) - \delta^h)h_t - L_t(1+r^L+\tau^L) & \text{if } d_t^h = B, \end{cases} \tag{42}$$

where

- p^h is the house price,
- τ^h is the housing transaction cost rate,
- δ^h is depreciation cost rate,
- $f_t(L_t)$ is the mortgage repayment (see below),
- r^L is the interest rate on mortgages,
- τ^L is mortgage transaction cost rate.

We follow Kaplan et al. (2019) and assume that mortgages are repaid according to the following function

$$f_t(L_t) = L_t \frac{r^L\left(1+r^L\right)^{T-t}}{\left(1+r^L\right)^{T-t} - 1}. \tag{43}$$

The rental rate is r^h. The required funds from the household block therefore is

$$x_t^h = \begin{cases} r^h \tilde{h}_t & \text{if } d_t^h = R \\ 0 & \text{if } d_t^h = O \\ p^h h_{t+1} - L_{t+1} & \text{if } d_t^h = B \end{cases} . \tag{44}$$

The law of motion for the housing stock and mortgage balance reflects the discrete choice

$$h_{t+1} = \begin{cases} 0 & \text{if } d_t^h = R \\ h_t & \text{if } d_t^h = O \\ \text{choice} & \text{if } d_t^h = B \end{cases} . \tag{45}$$

$$L_{t+1} = \begin{cases} 0 & \text{if } d_t^h = R \\ L_t(1 + r^L) - f_t(L_t) & \text{if } d_t^h = O \\ \text{choice} & \text{if } d_t^h = B \end{cases}$$

The mortgage choice is subject to a Loan-to-Value (LTV) constraint

$$0 \leq L_{t+1} \leq \kappa p^h h_{t+1},$$

which must be met only when a household is buying or refinancing. Owners who do not adjust may carry forward an L_t above the ceiling if house prices fall.

The utility flow of housing is

$$\tilde{h}_t = \begin{cases} \text{choice} & \text{if } d_t^h = R \\ h_{t+1} & \text{if } d_t^h = \{O, B\} \end{cases} . \tag{46}$$

Given the Cobb-Douglas specification of the utility function, the intratemporal first-order condition on housing for renters implies that

$$\tilde{h}_t = \frac{1-\alpha}{r^h} \frac{c_t}{\alpha} \quad \text{if } d_t^h = R. \tag{47}$$

(iii) Portfolio block. The portfolio block describes the accumulation of risky assets a_t and safe assets b_t. The safe asset yields a constant return r^b, while the risky asset

yields r_t^a drawn from an exogenous process. At the start of the period the household decides whether to keep their existing risky assets ($d_t^p = K$) or to adjust it ($d_t^p = A$). Adjusting incurs a fixed cost F^{adj}. The cash flow from this block is

$$m_t^p = \begin{cases} b_t & d_t^p = K, \\ b_t + a_t - F^{\text{adj}} & d_t^p = A. \end{cases} \tag{48}$$

while the cash out-flow from this block is

$$x_t^p = \begin{cases} \bar{b}_t & d_t^p = K, \\ \bar{b}_t + \bar{a}_t - F^{\text{adj}} & d_t^p = A. \end{cases}$$

and the stock of risky and safe assets are

$$\bar{a}_t = \begin{cases} a_t & d_t^p = K, \\ \text{choice} & d_t^p = A. \end{cases}, \tag{49}$$

$$\bar{b}_t = \text{choice}. \tag{50}$$

Short sales and borrowing are ruled out, so that $\bar{a}_t, \bar{b}_t \geq 0$.

4.2 Implementation

Policy network outputs. Instead of choosing the *levels* of the choice variables, we implement the continuous choices in the following way: Agents choose the savings rate λ_t^s, the share of risky assets λ_t^a, the share of expenses spent on housing λ_t^h, and the mortgage leverage λ_t^L, defined as:

$$\begin{aligned} \bar{a}_t + \bar{b}_t &= \lambda_t^s m_t, \\ \bar{a}_t &= \lambda_t^a \lambda_t^s m_t, \\ x_t^h &= (1 - \lambda_t^s) \lambda_t^h m_t, \\ \bar{L}_t &= \lambda_t^L \kappa p^h h_{t+1}. \end{aligned}$$

We then use the sigmoid function as the final activation function for all variables, ensuring that all outputs are between 0 and 1. This implementation ensures that all choices outputted by the policy network will be feasible.

Network architecture. Both the policy and the value networks have an architecture composed of two hidden layers of 500 neurons each. Activation functions for the output layer of the policy networks are set to sigmoid, to ensure that all the continuous choices are between 0 and 1. We train three distinct value networks during training, and use the average over the 3 during training and simulation. This allows us to reduce the noise induced when training the value function, which could propagate to both the continuous and discrete policy-functions during training. Learning rates are set to 10^{-4} for the policy network and 10^{-3} for the value networks, both decaying by a factor of 0.9999 per iteration with minimum rates of 10^{-5}.

Numerical integration. We use the Gauss-Hermite quadrature method with 5 nodes per continuous shocks to approximate the expectation operator. Combined with the two discrete shocks (job-finding and job-loss), this yields $2 \times 2 \times 5 \times 5 \times 5 = 250$ quadrature nodes per period.

Training and validation sample. We simulate a training sample of $N = 150$ agents in each iteration and maintain a replay buffer with memory size 8, implying a total buffer capacity of 1200 agents. The batch size used in the stochastic gradient updates equals the size of the contemporaneous training sample. Using a relatively small training sample reduces the cost of each episode, allowing the algorithm to run through more episodes per unit of wall-clock time. In combination with the replay buffer, this allows exploring a larger part of the state-space, especially early-on, when the policies used in simulation are very imprecise.

We use a validation sample of size $N = 100,000$, used to compute the average lifetime reward of policies over the training time. At the end of training, we recompute the average lifetime rewards for $N^{reps} = 10$ different validation samples.

Exploration. Exploration is introduced through two channels. For continuous actions (savings, portfolio shares, housing investment, and leverage), we add exploration-noise to the policy output with standard deviation $\sigma^{a} = 0.25$. For discrete choices, exploration is implemented by scaling the standard deviation of the Gumbel taste shocks by a factor 1.05 during training. Both mechanisms increase coverage of the effective state-action space.

Parameter	Value	Description	Source
Householdspreferences			
T	20	Maximum-lifetime	
σ	2	CRRA	
β	0.90	Discount factor	
α	0.89	Cobb-Douglas consumption preference share	
Jobladder			
π^e	0.20	Job-finding probability	
δ^e	0.05	Job-loss probability	
σ^w	0.20	Std dev.of job offers	
χ^u	0.10	Unemployment-benefits parameters	
$\bar{u}$	0.30	Maximum unemployment-benefits	
μ^{pe}, μ^{pu}	(0.0035, -0.0035)	Mean human capital accumulation	
$\underline{p}$	0.10	Lower bound on human capital	
Portfolio			
σ^r	0.16	Std.dev. of risky returns	
$\kappa = \mathbb{E}[r_t^a] - r^b$	0.03	Equity premium	
F	0.05	Fixed cost of portfolio choice	
Housing			
τ^h	0.05	Adjustment cost on housing	
τ^L	0.05	Adjustment cost on mortgages	
κ^h	0.90	1-downpayment share	
r^m	0.08	Mortgage interest rate	
ϕ	0.06	Rent-to-price ratio	

Table 3: Calibration

Backward. We perform a backward step of 3 hours to check the accuracy of our training, with a training sample of size $N = 100,000$. Each value and policy network per period has two hidden layers of 50 neurons each.

Hardware. We run the algorithm for 12 hours on an H100 GPU with 80 GB of RAM. Appendix TableC.1 reports all the relevant hyperparameters used.

4.3 Calibration and Results

Life cycle profile. Figure 9 reports the life-cycle profile of this model with $N =$ 100,000 households over $T = 20$ periods. The main life-cycle patterns follow the classic shape observed in this class of models: consumption increases steadily over time while the savings rate follows a hump-shape. Although the model does not feature explicit retirement, households endogenously exit the labor market as they age and accumulate enough financial and real assets. Wages and human capital also follow this hump-shape pattern. Average wages decline and human capital is gradually disaccumulated when households start leaving the labor market.

Portfolio dynamics exhibit a rebalancing motive: households accumulate relatively more risky assets in mid-life, before reallocating towards safe assets towards the end of their life. This is consistent with most typical financial advice to reallocate towards safe assets as households reach their retirement age where they will consume most of their wealth. Housing choices are also hump-shaped: the homeownership rate peaks in mid-life, leverage remains modest throughout, and the per-period purchase rate (share of buyers) is relatively high.

Accuracy. Figure 10 plots average log-10 Euler errors. They cluster around -2, with a heavier left-tail, indicating that our solution is accurate relative to this first order condition. Note that, in our current implementation of the algorithm, we haven't targeted the Euler error explicitly.

Figure 11 shows the total expected reward over computation time, expressed as transfer equivalents relative to the final solution. We compute the transfer that makes a household indifferent between the solution available at each point in computation time and the final converged solution. We finally perform a backward iteration for 180 minutes, starting from the converged solution, and compare the results. We find that it does not find a better solution, suggesting that our algorithm has converged and found an approximately optimal solution.

Finally, Figure 12 plots some moments of the simulated sample over the training time. We compute the mean of consumption (share of discrete choices) over agents and time using our validation sample, and plot it at each iteration. We find that the solution stabilizes over training time and does not significantly change in the last hours of training time.

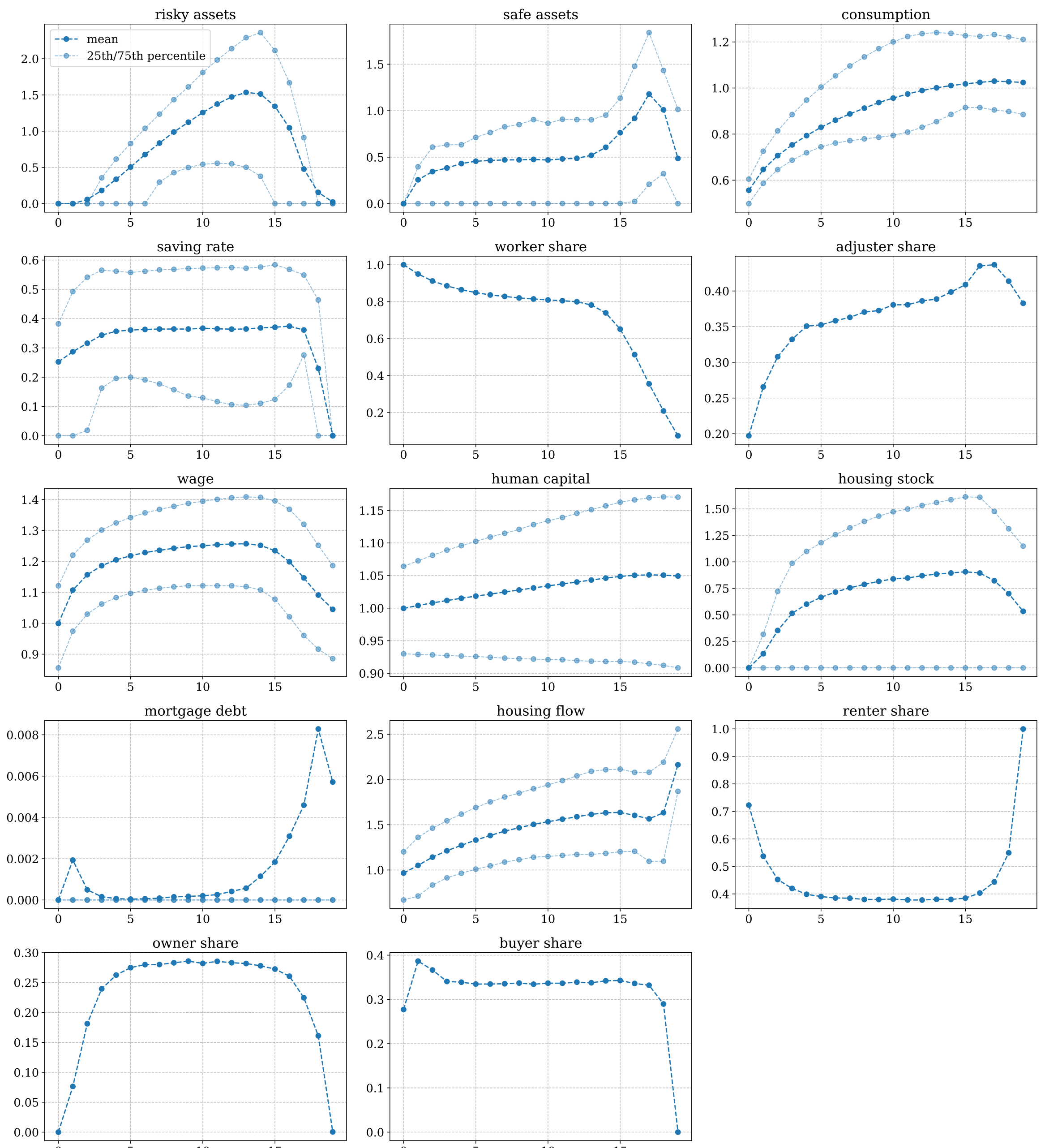

Figure 9: Large-Scale Life Cycle Model: Life cycles profiles

Notes: Each panel shows the average value in the sample, per period, of a given variable. The upper shaded line shows the value for the top 75th percentile, while the bottom line shows the corresponding value for the bottom 25th percentile.

var	moment	$t = 1$	$t = 9$	$t = 14$	$t = 19$
a	Mean	0.000	1.124	1.514	0.023
	Median	0.000	1.055	1.358	0.000
	Variance	0.000	0.795	1.685	0.029
	Skewness	61.064	1.024	0.789	8.385
	Kurtosis	5115.279	5.384	3.606	80.901
b	Mean	0.257	0.476	0.606	0.486
	Median	0.276	0.225	0.243	0.189
	Variance	0.039	0.335	0.690	0.290
	Skewness	0.165	1.318	2.250	0.538
	Kurtosis	2.309	4.655	10.267	1.755
c	Mean	0.647	0.937	1.011	1.024
	Median	0.654	1.020	1.112	1.042
	Variance	0.021	0.116	0.119	0.110
	Skewness	-1.046	-0.839	-1.049	-0.428
	Kurtosis	5.996	2.989	3.490	4.690
h	Mean	0.134	0.816	0.895	0.533
	Median	0.000	1.038	1.071	0.000
	Variance	0.052	0.502	0.622	0.393
	Skewness	1.368	-0.011	0.057	0.522
	Kurtosis	3.509	1.447	1.498	1.655
m	Mean	0.002	0.000	0.001	0.006
	Median	0.000	0.000	0.000	0.000
	Variance	0.000	0.000	0.001	0.003
	Skewness	8.776	52.877	27.760	11.510
	Kurtosis	88.465	3512.594	899.486	155.262
w	Mean	1.107	1.248	1.252	1.045
	Median	1.089	1.260	1.275	1.027
	Variance	0.036	0.050	0.057	0.047
	Skewness	0.616	-0.128	-0.213	0.472
	Kurtosis	3.784	3.430	3.093	3.253
p	Mean	1.004	1.031	1.046	1.049
	Median	0.999	1.020	1.031	1.031
	Variance	0.012	0.024	0.033	0.040
	Skewness	0.315	0.455	0.517	0.574
	Kurtosis	3.197	3.409	3.480	3.615

Table 4: Main Moments of the Large Model

Correlation	$t=1$	$t=9$	$t=14$	$t=19$
Corr(a,b)	0.079	0.012	0.090	-0.096
Corr(a,c)	0.058	0.661	0.634	0.122
Corr(a,h)	-0.014	0.175	0.172	-0.058
Corr(a,m)	-0.003	0.013	0.090	0.416
Corr(a,w)	0.081	0.496	0.455	-0.010
Corr(a,p)	0.018	0.299	0.303	0.001
Corr(b,c)	0.115	0.110	0.177	0.178
Corr(b,h)	-0.762	-0.614	-0.327	-0.735
Corr(b,m)	-0.179	-0.012	-0.010	-0.067
Corr(b,w)	0.182	0.075	0.013	0.020
Corr(b,p)	0.078	0.089	0.235	0.092
Corr(c,h)	0.238	0.409	0.405	0.350
Corr(c,m)	-0.004	-0.027	0.000	0.064
Corr(c,w)	0.561	0.686	0.632	0.127
Corr(c,p)	0.469	0.382	0.376	0.308
Corr(h,m)	0.175	-0.015	-0.005	0.055
Corr(h,w)	0.197	0.279	0.263	0.019
Corr(h,p)	0.107	0.144	0.182	0.118
Corr(m,w)	-0.057	-0.031	-0.056	-0.007
Corr(m,p)	0.063	0.013	0.032	-0.000
Corr(w,p)	0.003	0.036	0.006	-0.086

Table 5: Main Correlations of the Large Model

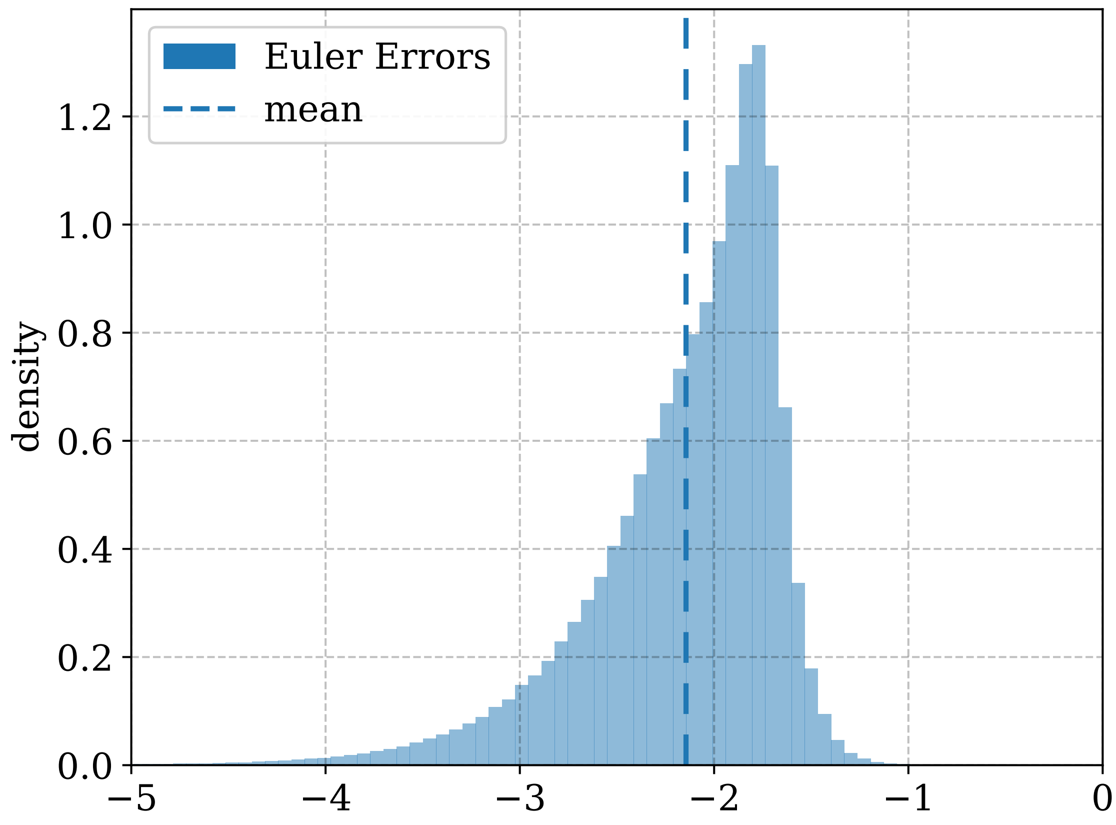

Figure 10: Large-Scale Life Cycle Model: Euler errors (log 10)

Notes: This figure shows the distribution of Euler Errors for unconstrained agents, bundled for all periods. See Figure 2 for details.

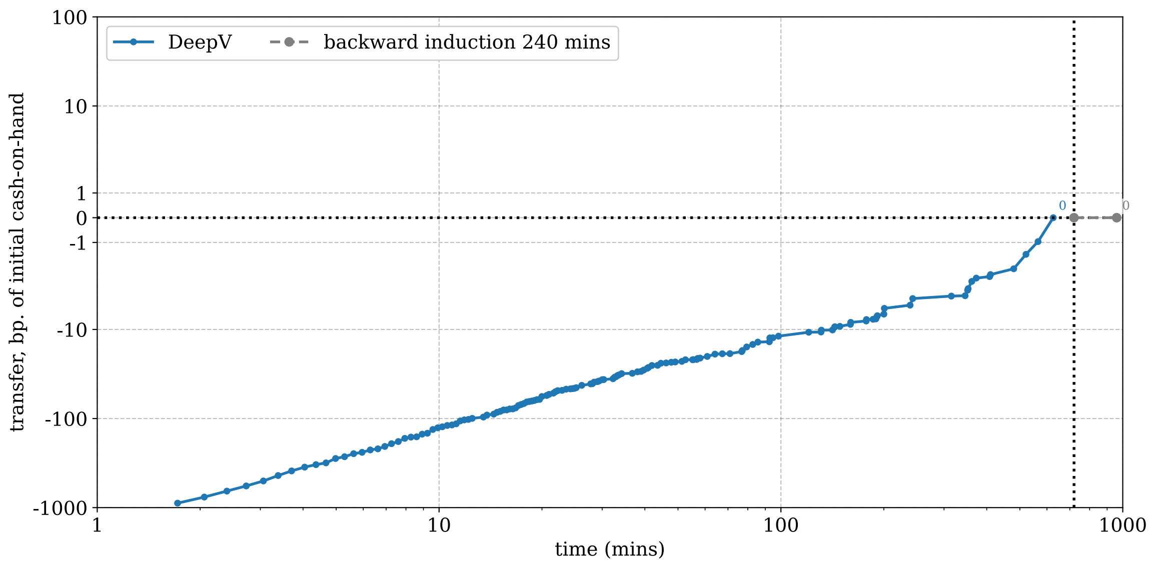

Figure 11: Large-Scale Life Cycle Model: Average Lifetime Rewards

Notes: The x-axis shows time in minutes, while the y-axis shows the transfer required to make agents indifferent between the final solution and the current DL solution. When below 0, this means that the agent would have demanded a transfer of x units of cash-on-hand to use the current solution instead of the final one. We only show the transfer when the algorithm improves.

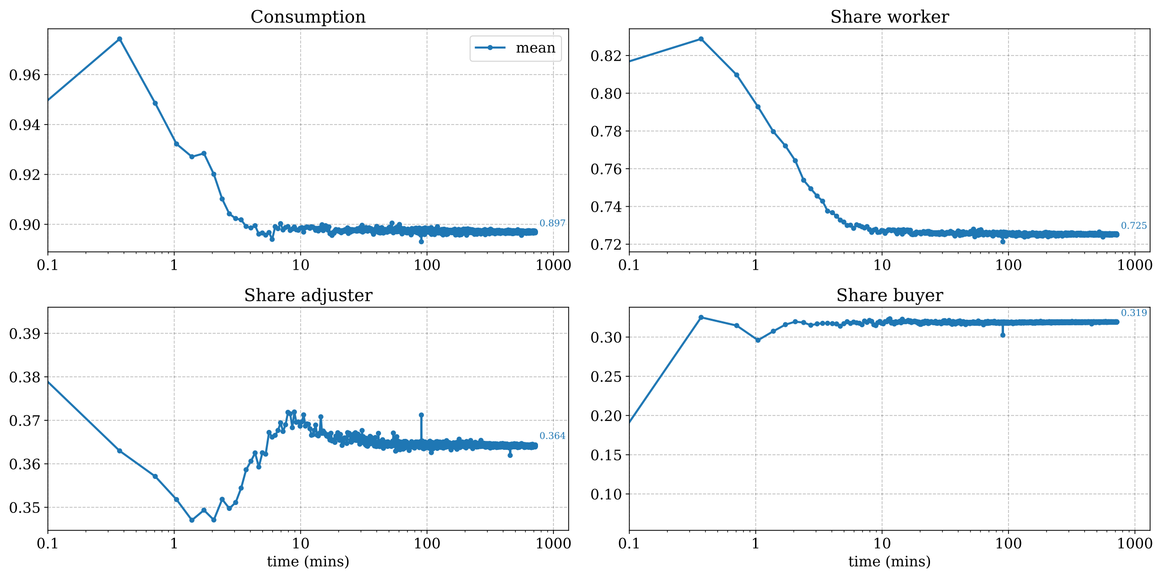

Figure 12: Large-Scale Life Cycle Model: Moments over Training Time

Notes: The x-axis shows training time in minutes, while the y-axis plots the mean of consumption in the sample, over the entire lifetime, for the first panel, and the share of worker / adjuster / buyer, for the remaining one. A stabilizing line indicates that the solution does not meaningfully change over training time.

5 Conclusion

This paper develops a deep learning solution method, DeepV, for finite-horizon dynamic models with non-convexities and discrete-continuous choices, combining a post-decision value network with a continuous-policy network trained via simulation. We test our algorithm on two models: a durable-goods model with non-convexities, and a large-scale life cycle model with a job-ladder, a portfolio block, and a housing block. The approach matches dynamic programming in accuracy while becoming relatively more efficient as dimensionality increases, and scales to a large life-cycle model with employment, housing, and portfolio blocks that delivers plausible profiles and low Euler-errors. We also provide a transparent python library to reproduce our results and solve general life-cycle models with non-convexities.

References

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

Azinovic, M. and Žemlička, J. (2023). Intergenerational Consequences of Rare Disasters. Working Paper.

Azinovic-Yang, M. and Žemlička, J. (2025). Deep Learning in the Sequence Space. Working Paper.

Brumm, J., Krause, C., Schaab, A., and Scheidegger, S. (2021). Sparse grids for dynamic economic models. *SSRN Electronic Journal*.

Brumm, J. and Scheidegger, S. (2017). Using Adaptive Sparse Grids to Solve High-Dimensional Dynamic Models. *Econometrica*, 85(5):1575–1612.

Druedahl, J. and Røpke, J. (2026). Deep Learning Algorithms for Solving Convex Finite-Horizon Models. Working Paper.

Duarte, V., Fonseca, J., Goodman, A. S., and Parker, J. A. (2022). Simple Allocation Rules and Optimal Portfolio Choice Over the Lifecycle. Working Paper 29559, National Bureau of Economic Research.

Fernández-Villaverde, J., Nuño, G., and Perla, J. (2024). Taming the Curse of Dimensionality: Quantitative Economics with Deep Learning. Technical Report w33117, National Bureau of Economic Research, Cambridge, MA.

Gomes, F. (2020). Portfolio Choice Over the Life Cycle: A Survey. *Annual Review of Financial Economics*, 12(Volume 12, 2020):277–304. Publisher: Annual Reviews.

Gourinchas, P.-O. and Parker, J. A. (2002). Consumption over the life cycle. *Econometrica*, 70(1):47–89.

Gu, Z., Laurière, M., Merkel, S., and Payne, J. (2024). Deep Learning Solutions to Master Equations for Continuous Time Heterogeneous Agent Macroeconomic Models. Working Paper.

Han, J., Yang, Y., and E, W. (2025). DeepHAM: A Global Solution Method for Heterogeneous Agent Models with Aggregate Shocks. Working Paper.

Heathcote, J., Storesletten, K., and Violante, G. L. (2009). Quantitative Macroeconomics with Heterogeneous Households. *Annual Review of Economics*, 1(1):319–354.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Judd, K. L., Maliar, L., and Maliar, S. (2017). How to Solve Dynamic Stochastic Models Computing Expectations Just Once. *Quantitative Economics*, 8(3).

Kaplan, G., Mitman, K., and Violante, G. L. (2019). The Housing Boom and Bust: Model Meets Evidence. *Journal of Political Economy*, page 89.

Kase, H., Melosi, L., and Rottner, M. (2024). Estimating Nonlinear Heterogeneous Agents Models with Neural Networks. Working Paper.

Maliar, L. and Maliar, S. (2022). Deep learning classification: Modeling discrete labor choice. *Journal of Economic Dynamics and Control*, 135:104295.

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

Modigliani, F. and Brumburg, R. (1954). Utility Analysis and the Consumptio Function: An Interpretation of Cross-Section Data. In Kurihara, K. and Brunswick, N., editors, *Post-Keynesian Economics*, pages 338–436. Rutgers University Press.

Nardi, M. D., French, E., and Jones, J. B. (2016). Savings After Retirement: A Survey. *Annual Review of Economics*, 8(Volume 8, 2016):177–204. Publisher: Annual Reviews.

Pascal, J. (2024). Artificial neural networks to solve dynamic programming problems: A bias-corrected Monte Carlo operator. *Journal of Economic Dynamics and Control*, 162:104853.

Payne, J., Rebei, A., and Yang, Y. (2024). Deep Learning for Search and Matching Models. SSNN.

Scheidegger, S. and Bilionis, I. (2019). Machine learning for high-dimensional dynamic stochastic economies. *Journal of Computational Science*, 33:68–82.

Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., and Schmidhuber, J. (2010). Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559.

A Algorithm

A.1 Hyper-parameters

This appendix describes the main hyper-parameters of the algorithm.

Training sample, validation sample, and replay buffer.

1. Simulation sample of size N^{simul} (also used as validation sample)
2. Training sample size N
3. Buffer memory: Replay buffer memory size.
4. Batch size: Size of the batch used for the ADAM optimizer

Neural network architecture.

1. Number of hidden layers and neurons for the policy network.
2. Number of hidden layers and neurons for the value network.
3. Number of value networks to train.
4. Final activation function for the policy function,

Learning rates.

1. Policy function learning rate, decay, and minimum value.
2. Value function learning, decay, and minimum value .
3. τ: update coefficient for the targets networks.

Exploration.

1. σ^{ε^a} : variance of exploration noise for continuous variables (along with potential decay and minimum value).
2. σ^{ε^d} : scaling parameter for the taste-shock, used as exploration during simulation.

Convergence & Epochs.

1. Δ_R: frequency of large simulation evaluation using the validation sample.
2. K_{time} : minimum training time .
3. $\#_\pi, \#_{\overline{v}}$: number of epochs for policy and value update .

A.2 First order conditions and Multiple Value Networks

We can extend our solution algorithm to include information from first order conditions and allow for multiple value networks. To do so, we replace the Algorithms 2 and 3 with the Algorithms A.2 and A.3.

Targeting first-order conditions. We extend the value network to be approximating both the level of the post-decision value function and selected derivatives. We assume that some relevant first order conditions in vector form can be written as

$$g\left(s_t, a_t, \overline{q}_t\right) = 0 \tag{51}$$

where $\overline{q}_t = \overline{q}(\bar{s}_t)$ contains the relevant post-decision marginal values. We define $\overline{q}$ and $\overline{q}_h$ as

$$\overline{q}(\bar{s}_t) = \begin{cases} \overline{q}_h(\bar{s}_t) & \text{if } t = T-1 \\ \mathbb{E}_t\left[q(t+1, s_{t+1}, a_{t+1})\right] & \text{else} \end{cases} \tag{52}$$

where $q(t+1, s_{t+1}, a_{t+1})$ is the marginal reward. We denote by $dv(\bullet)$ the analytically derived derivative of the value function using the Envelope theorem.

Details are included in Algorithms A.2 and A.3. Importantly, we now compute both the level of the post-decision value and some of its derivatives. We then train on both the value-of-choice and squared errors in the selected first order condition. We introduce the positive weight parameters $\kappa_{\overline{q}} > 0$ and $\kappa_{\text{FOC}} > 0$. Note that, when $\#_{\overline{v}} = 1$ and $\kappa_{\overline{q}} = \kappa_{\text{FOC}} = 0$ we are back at the baseline DeepV algorithm from Section 2.

Multiple value networks. We implement a version of ensemble learning by training multiple value networks at once. We draw independent initial parameters for $\#_{\overline{v}}$ different value networks indexed by m. In Algorithm 3, we update the value networks independently in step 2, and in step 3, and use the average across value networks, i.e.

$$\overline{v}\left(t, \tilde{\bar{s}}_{t,i}; \theta_{\overline{v}}^{k}\right) = \frac{1}{\#_{\overline{v}}} \sum_{m=0}^{\#_{\overline{v}}-1} \overline{v}\left(t, \tilde{\bar{s}}_{t,i}; \theta_{\overline{v}}^{m,k}\right). \tag{53}$$

All value networks receive the same data batches, so that the only randomness averaged-out comes from the different initialization of each network. This turns out to improve performance.

Algorithm A.2 Value and marginal value target

Given post-decision state $\bar{s}_{t,i}$ and network parameters $\theta_{\overline{v}}$ and θ_{π^a} compute

$$\tilde{\bar{v}}_{t,i} = \sigma_\varepsilon \log \left[\sum_{j \in \{0,1,\dots,\#_d-1\}} \exp\left(\tilde{\bar{v}}^j_{t,i} / \sigma_\varepsilon \right) \right] \tag{54}$$

$$\tilde{\bar{q}}_{t,i} = \mathbb{E}^z_t \left[\sum_{j \in \{0,1,\dots,\#_d-1\}} \tilde{p}^j_{t,i} \cdot dv\left(\tilde{\bar{s}}^j_{t+1}, \tilde{a}^j_{t+1,i}, \tilde{d}^j_{t+1} \right) \right] \tag{55}$$

where

$$\tilde{\bar{v}}^j_{t,i} = \mathbb{E}^z_t \left[u_{t+1}\left(\tilde{s}_{t+1,i}, \tilde{d}^j_{t+1}, \tilde{a}^j_{t+1,i} \right) + \beta \begin{cases} h(\tilde{\bar{s}}_{t+1,i}) & \text{if } t = T-2 \\ \overline{v}\left(t+1, \tilde{\bar{s}}_{t+1,i}; \theta_{\overline{v}}\right) & \text{else} \end{cases} \right] \tag{56}$$

$$\tilde{p}^j_{t,i} = \mathbb{E}^z_t \left[\frac{\exp\left(\tilde{\bar{v}}^j_{t,i} / \sigma_\varepsilon \right)}{\sum_{k=0}^{\#_d-1} \exp\left(\tilde{\bar{v}}^k_{t,i} / \sigma_\varepsilon \right)} \right] \tag{57}$$

for $j \in \{0, 1, \dots, \#_d - 1\}$ where

$$\tilde{z}_{t+1,i} \sim F^z_{t+1}\left(\bar{s}_{t,i} \right) \tag{58}$$

$$\tilde{s}_{t+1,i} = \Gamma_t(\bar{s}_{t,i}, \tilde{z}_{t+1,i}) \tag{59}$$

$$\left(\tilde{a}^0_{t+1,i}, \tilde{a}^1_{t+1,i}, \dots, \tilde{a}^{\#_d-1}_{t+1,i} \right) = \pi^a(t+1, \tilde{s}_{t+1,i}, \tilde{d}^j_{t+1,i}; \theta_{\pi^a}) \tag{60}$$

$$\tilde{\bar{s}}_{t+1,i} = \overline{\Gamma}_{t+1}(\tilde{s}_{t+1,i}, \tilde{a}^j_{t+1,i}, \tilde{d}^j_{t+1}). \tag{61}$$

Algorithm A.3 Updating parameters with FOC and multiple value networks

If $k = 0$, set target network parameters $\check{\theta}^{-1}_{\overline{v},m} = \theta^{-1}_{\overline{v},m}$ for $m \in \{0, \dots, \#_{\overline{v}} - 1\}$ and $\check{\theta}^{-1}_{\pi^a} = \theta^{-1}_{\pi^a}$.

1. Compute the value targets $\tilde{\overline{v}}_{t,i,m}$ and marginal value targets $\tilde{\overline{q}}_{t,i,m}$ for all $\bar{s}_{t,i} \in \mathcal{B}^k$ with $t < T - 1$ using Algorithm A.2 with the lagged target network parameters, $\check{\theta}^{k-1}_{\pi^a}$ and $\check{\theta}^{k-1}_{\overline{v},m}$.
2. For each $m \in \{0, \dots, \#_{\overline{v}} - 1\}$: Update the value network to $\theta^k_{\overline{v},m}$ for $\#_{\overline{v}}$ epochs using the loss function

$$L_{\overline{v}\overline{q}}\left(\theta_{\overline{v},m}; \mathcal{B}^k\right) = \frac{1}{|\mathcal{B}^k|} \sum_{i,t,\bar{s}_{t,i} \in \mathcal{B}^k} \left[\left(\overline{v}\left(t, \bar{s}_{t,i}; \theta_{\overline{v},m}\right) - \tilde{\overline{v}}_{t,i,m}\right)^2 + \kappa_{\overline{q}} \left(\overline{q}\left(t, \bar{s}_{t,i}; \theta_{\overline{v},m}\right) - \tilde{\overline{q}}_{t,i,m}\right)^2 \right] \tag{62}$$

   Update the value target parameters: $\check{\theta}^k_{\overline{v},m} = \tau \theta^k_{\overline{v},m} + (1-\tau)\check{\theta}^{k-1}_{\overline{v},m}$.

3. If $k > \underline{k}_{\pi^a}$: Update the policy network to $\theta^k_{\pi^a}$ for $\#_\pi$ epochs using the loss function

$$L_{\pi^a}(\theta_{\pi^a}; \mathcal{B}^k) = -\frac{1}{|\mathcal{B}^k| \cdot \#_d} \sum_{i,t,s_{t,i} \in \mathcal{B}^k} \sum_{d^j_{t,i} \in \mathcal{D}_t(s_{t,i})} v_{t,i,j} + \kappa_{\text{FOC}} \cdot \text{FOC}_{t,i,j} \tag{63}$$

   where

$$v_{t,i,j} = u_t(s_{t,i}, \tilde{a}^j_{t,i}, d^j) + \beta \begin{cases} h(\tilde{\bar{s}}_{t,i}) & \text{if } t = T-1 \\ \overline{v}\left(t, \tilde{\bar{s}}_{t,i}; \theta^k_{\overline{v}}\right) & \text{else} \end{cases} \tag{64}$$

$$\text{FOC}_{t,i,j} = \begin{cases} g\left(s_{t,i}, \tilde{a}^j_{t,i}, \tilde{\bar{s}}_{t,i}, \overline{q}\left(t, \tilde{\bar{s}}_{t,i}; \theta^k_{\overline{v}}\right)\right)^2 & \text{if } t < T-1 \\ g_h\left(s_{t,i}, \tilde{a}^j_{t,i}, \tilde{\bar{s}}_{t,i}\right)^2 & \text{if } t = T-1 \end{cases} \tag{65}$$

$$\left(\tilde{a}^0_{t,i}, \tilde{a}^1_{t,i}, \dots, \tilde{a}^{\#_d - 1}_{t,i}\right) = \pi^a(t, s_{t,i}; \theta^k_{\pi^a}) \tag{66}$$

$$\tilde{\bar{s}}_{t,i} = \overline{\Gamma}_t(s_{t,i}, \tilde{a}^j_{t,i}, d^j_{t,i}) \tag{67}$$

$$\overline{v}\left(t, \tilde{\bar{s}}_{t,i}; \theta^k_{\overline{v}}\right) = \frac{1}{\#_{\overline{v}}} \sum_{m=0}^{\#_{\overline{v}}-1} \overline{v}\left(t, \tilde{\bar{s}}_{t,i}; \theta^k_{\overline{v},m}\right) \tag{68}$$

$$\overline{q}\left(t, \tilde{\bar{s}}_{t,i}; \theta^k_{\overline{v}}\right) = \frac{1}{\#_{\overline{v}}} \sum_{m=0}^{\#_{\overline{v}}-1} \overline{q}\left(t, \tilde{\bar{s}}_{t,i}; \theta^k_{\overline{v},m}\right). \tag{69}$$

   Update the target policy parameters: $\check{\theta}^k_{\pi^a} = \tau \theta^k_{\pi^a} + (1-\tau)\check{\theta}^{k-1}_{\pi^a}$.

Allowing for constraints. For the consumption-savings problem with a borrowing constraint $\bar{m}_t \geq 0$, the first order condition combines the Euler equation with complementary slackness. One solution is to explicitly approximate the Lagrange multiplier as an output of the policy network. We take another route and use instead the Fischer-Burmeister function $\phi_{\text{FB}}(a,b) = \sqrt{a^2+b^2} - a - b$ which satisfies $\phi_{\text{FB}}(a,b) = 0$ if and only if $a \geq 0$, $b \geq 0$, and $ab = 0$. The FOC becomes

$$g\left(s_t, a_t, \bar{s}_t, \overline{q}_t\right) = \phi_{\text{FB}}\left(\frac{u'(c_t)}{\beta \cdot \overline{q}_t} - 1, \bar{m}_t\right) \tag{70}$$

where $\overline{q}_t = \partial \overline{v}_t / \partial \bar{m}_t$ is the marginal value of post-decision wealth. This formulation ensures that either the Euler equation holds with equality ($u'(c_t) = \beta \cdot \overline{q}_t$), or the borrowing constraint binds ($\bar{m}_t = 0$).

For the terminal period with no bequest motive, the optimal policy is to consume all wealth, so the terminal FOC simplifies to

$$g_h\left(s_{T-1}, a_{T-1}, \bar{s}_{T-1}\right) = \bar{m}_{T-1}. \tag{71}$$

The marginal reward for non-terminal periods is given by $q(s_t, d_t, a_t) = R \cdot u'(c_t)$, and the terminal marginal value is $\overline{q}_h(\bar{s}_{T-1}) = 0$.

A.3 Canonical buffer-stock model

The problem is fully described by the Bellman equation below:

$$
\begin{aligned}
v_t(m_t, p_t) &= \max_{c_t} u_t(c_t) + \beta \mathbb{E}_t \left[v_{t+1}(m_{t+1}, p_{t+1}) \right] \qquad (72)\\
&\text{s.t.}\\
a_t &= 1 - \frac{c_t}{m_t}\\
\overline{m}_t &= a_t m_t\\
m_{t+1} &= (1+r)\overline{m}_t + y_{t+1}(p_{t+1}, \psi_{t+1})\\
p_{t+1} &= p_t^{\rho_p} \xi_{t+1}, \;\; \xi_{t+1} \sim F^{\xi}_{t+1} = \exp \mathcal{N}(-0.5\sigma^2_{\xi}, \sigma^2_{\xi}),\\
y_{t+1}(p_{t+1}, \psi_{t+1}) &= \begin{cases} \kappa_t & \text{if } t \geq T^{\text{retired}} \\ \kappa_t \psi_{t+1} p_{t+1} & \text{else} \end{cases}\\
\overline{m}_t \;\; \psi_{t+1} &\sim F^{\psi}_{t+1} = \exp \mathcal{N}(-0.5\sigma^2_{\psi}, \sigma^2_{\psi}), \geq 0
\end{aligned}
$$

where m_t is cash-on-hand, p_t is persistent income, a_t is a savings-rate, $\overline{m}_t$ is post-decision cash-on-hand, y_{t+1} is income, ψ_{t+1} is a transitory shock and ξ_{t+1} is a persistent shock. κ_t describes the life-cycle aspect of income.

In Druedahl and Røpke (2026), it is shown that this model can be solved efficiently with an algorithm DeepFOC, which uses first-order conditions to solve the problem. This algorithm cannot be applied to non-convex models, where first order conditions are not sufficient.

Figure A.2 shows speed and accuracy of DeepV with one value network (orange line), DeepV with 3 value neural networks (green line), DeepV using information from FOCs (red line), DeepV using 3 value networks and FOCs (purple line). Everything is compared to an EGM solution of the Buffer-stock model. The EGM solution solves this problem in less than a second making it significantly faster than any DL algorithm. We also show the performance of the DeepFOC algorithm in this problem (blue line). Using FOCs and multiple value networks significantly improve the accuracy of DeepV. DeepFOC is faster and more precise than all versions of DeepV.

Figure A.3 shows speed and accuracy for the same model, but where we compare DeepV with one, three and five value networks against the EGM solution. While going to three value networks improve precision going to 5 has little extra effect.

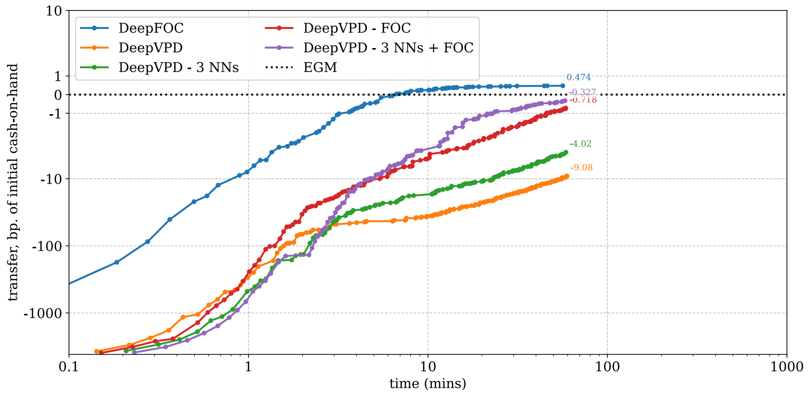

Figure A.2: DeepV for Buffer-stock model: Convergence

Notes: See description of Figure 1.

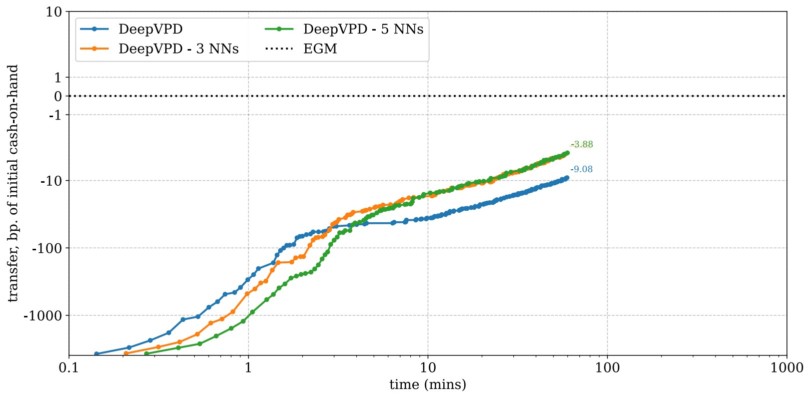

Figure A.3: DeepV for Buffer-stock model: Convergence when changing the number of trained value networks.

Notes: The x-axis shows time in minutes, while the y-axis shows the transfer required to make agents indifferent between the DP solution and the current DL solution. When below 0, this means that the agent would have demanded a transfer of x units of cash-on-hand to use the DL solution instead of the DP one. When above 0, it means that the agent would have demanded a transfer of x units of cash-on-hand to use the DP solution instead of the DL one.

A.4 Backward Induction

Solving the model simultaneously across time periods has a number of benefits in terms of exploiting symmetries in policy functions across time periods. Working on both sides of the Bellman equation might, however, cause instabilities. To check whether this is an issue we can also do a backward-induction step to verify the accuracy of the solution.

The idea is as follows: Consider having run DeepV for K episodes. Using the obtained policy and value networks, we can simulate a very large sample of states. Using that sample we can then perform backward induction period-by-period to check whether this allows us to find a better solution. This requires a new algorithm: *DeepVBackward*, described in detail in Algorithm A.4.

After having applied the baseline DeepV we have parameters $\theta^*_{\pi^a}$ for the policy network and $\theta^*_{\overline{v}}$ for the value network. We now work with T independent policy networks and T independent value networks. Let $\underline{\pi}^a(t, s_t; \theta^*_{\pi^a})$ be the previous policy network before applying the final activation function f such that $\pi^a(t, s_t; \theta^*_{\pi^a}) = f(\underline{\pi}^a(t, s_t; \theta^*_{\pi^a}))$. The period t policy function is now modeled as

$$\pi^a_t(s_t; \theta^*_{\pi^a}, \theta_{t,\pi}) = f\left(\underline{\pi}^a(t, s_t; \theta^*_{\pi^a}) + \underline{\pi}^a_t(s_t; \theta_{t,\pi})\right),$$

where $\underline{\pi}^a_t(s_t; \theta_{t,\pi})$ is the period t neural network with unknown parameters $\theta_{t,\pi}$. Similarly, for the post-decision value function, we model

$$\overline{v}_t(\overline{s}_t; \theta^*_{\overline{v}}, \theta_{t,\overline{v}}) = \overline{v}(t, \overline{s}_t; \theta^*_{\overline{v}}) + \overline{v}_t(\overline{s}_t; \theta_{t,\overline{v}}),$$

where $\overline{v}_t(\overline{s}_t; \theta_{t,\overline{v}})$ is the period t value correction network with unknown parameters $\theta_{t,\overline{v}}$. If the networks $\pi^a(t, s_t; \theta^*_{\pi^a})$ and $\overline{v}(t, \overline{s}_t; \theta^*_{\overline{v}})$ are the true policy and value functions (approximately), a backward induction step should learn that $\underline{\pi}^a_t(s_t; \theta_{t,\pi}) \approx 0$ and $\overline{v}_t(\overline{s}_t; \theta_{t,\overline{v}}) \approx 0$.

First, a large training sample is simulated using the previously obtained values of $\theta^*_{\pi^a}$ and $\theta^*_{\overline{v}}$. Second, we loop backwards through all the periods. In each period, we first update the value network, and then update the policy network. For the value network, we go through the training sample for $\#_{\overline{v}}$ epochs, where we update the parameters in random batches, and continue with the parameters from the epoch which gave the best average loss. For the policy network, we similarly train for $\#_{\pi}$

epochs.

We initialize $\theta_{t,\pi}$ and $\theta_{t,\overline{v}}$ using $\theta_{t+1,\pi}$ and $\theta_{t+1,\overline{v}}$ respectively, except in the last period, where the parameters are initialized such that $\underline{\pi}^a_{T-1}(s_{T-1};\theta_{T-1,\pi}) \approx 0$ and $\overline{v}_{T-1}(\overline{s}_{T-1};\theta_{T-1,\overline{v}}) \approx 0$, so the initial behavior is the one implied by the previous networks.

Algorithm A.4 DeepVBackward

```
Draw training sample $\mathcal{S}$ using previously obtained $\theta^*_{\pi^a}$ and $\theta^*_{\overline{v}}$.
Initialize $\theta_{T-1,\pi}$ and $\theta_{T-1,\overline{v}}$ so $\underline{\pi}^a_{T-1}(s_{T-1};\theta_{T-1,\pi}) \approx 0$ and $\overline{v}_{T-1}(\overline{s}_{T-1};\theta_{T-1,\overline{v}}) \approx 0$.
For $t \in \{T-1, T-2, \dots, 0\}$ do:
  1. **Update value network** using Algorithm A.5.
  2. **Update policy network** using Algorithm A.6.
  3. If $t > 0$: Set $\theta_{t-1,\pi} = \theta_{t,\pi}$ and $\theta_{t-1,\overline{v}} = \theta_{t,\overline{v}}$.
Return $\theta_{0,\pi}, \theta_{1,\pi}, \dots, \theta_{T-1,\pi}$ and $\theta_{0,\overline{v}}, \theta_{1,\overline{v}}, \dots, \theta_{T-1,\overline{v}}$.
```

Algorithm A.5 DeepVBackward: Value network update

For $\#_{\overline{v}}$ epochs go through the training sample in random batches and update $\theta_{t,\overline{v}}$ using the loss function

$$L_{\overline{v}}\left(\theta_{t,\overline{v}};\mathcal{S}\right) = \frac{1}{N}\sum_{i,\overline{s}_{t,i}\in\mathcal{S}}\left(\overline{v}_t(\overline{s}_{t,i};\theta^*_{\overline{v}},\theta_{t,\overline{v}}) - \tilde{\overline{v}}_{t,i}\right)^2$$

where the value target is

$$\begin{aligned}
\tilde{\overline{v}}_{t,i} &= \sigma_\varepsilon \log\left[\sum_{j\in\{0,1,\dots,\#_d-1\}} \exp\left(\tilde{\overline{v}}^j_{t,i}/\sigma_\varepsilon\right)\right] \\
\tilde{\overline{v}}^j_{t,i} &= \mathbb{E}^z_t\left[u_{t+1}\left(\tilde{s}_{t+1,i}, \tilde{d}^j_{t+1}, \tilde{a}^j_{t+1,i}\right) + \beta\begin{cases} h(\tilde{\overline{s}}_{t+1,i}) & \text{if } t = T-2 \\ \overline{v}_{t+1}(\tilde{\overline{s}}_{t+1,i};\theta^*_{\overline{v}},\theta_{t+1,\overline{v}}) & \text{else}\end{cases}\right]
\end{aligned}$$

and

$$\begin{aligned}
\tilde{s}_{t+1,i} &= \Gamma_t(\overline{s}_{t,i},\tilde{z}_{t+1,i}), \quad \tilde{z}_{t+1,i} \sim F^z_{t+1}\left(\overline{s}_{t,i}\right) \\
\left(\tilde{a}^0_{t+1,i}, \tilde{a}^1_{t+1,i}, \dots, \tilde{a}^{\#_d-1}_{t+1,i}\right) &= \pi^a_{t+1}(\tilde{s}_{t+1,i};\theta^*_{\pi^a},\theta_{t+1,\pi}) \\
\tilde{\overline{s}}_{t+1,i} &= \overline{\Gamma}_{t+1}(\tilde{s}_{t+1,i},\tilde{a}^j_{t+1,i},d^j_{t+1})
\end{aligned}$$

Continue with $\theta_{t,\overline{v}}$ from the epoch with the smallest value loss.

Algorithm A.6 DeepVBackward: Policy network update

For $\#_\pi$ epochs go through the training sample in random batches and update $\theta_{t,\pi}$ using the loss function

$$L_\pi\left(\theta_{t,\pi}; \mathcal{S}\right) = -\frac{1}{N \cdot \#_d} \sum_{i, s_{t,i} \in \mathcal{S}} \sum_{d_t^j \in \mathcal{D}_t(s_{t,i})} \tilde{\mathcal{V}}_{t,i,j}$$

where

$$\begin{aligned} \tilde{\mathcal{V}}_{t,i,j} &= u_t(s_{t,i}, d_t^j, \tilde{a}_{t,i}^j) + \beta \begin{cases} h(\tilde{\tilde{s}}_{t,i}) & \text{if } t = T-1 \\ \overline{v}_t(\tilde{\tilde{s}}_{t,i}; \theta_{\overline{v}}^*, \theta_{t,\overline{v}}) & \text{else} \end{cases} \\ \left(\tilde{a}_{t,i}^0, \tilde{a}_{t,i}^1, \ldots, \tilde{a}_{t,i}^{\#_d - 1}\right) &= \pi_t^a(s_{t,i}; \theta_{\pi^a}^*, \theta_{t,\pi}) \\ \tilde{\tilde{s}}_{t,i} &= \overline{\Gamma}_t(s_{t,i}, \tilde{a}_{t,i}^j, d_t^j) \end{aligned}$$

Continue with $\theta_{t,\pi}$ from the epoch with the smallest policy loss.

B Test Model

B.1 Calibration

- **Time:** $T = 20$,
- **Preferences:** $\beta = 0.965$, $\omega_1 = 0.10$, $\omega_2 = 0.10$,$\underline{d}_1 = \underline{d}_2 = 0.10$, $\rho = 2$.
- **Income process:** $\sigma_\psi = 0.1$, $\sigma_\xi = 0.1$,$\rho_p = 0.95$.
- **Assets market:** $r = 0.03$, $\kappa = 0.1$, $\delta = 0.15$.
- **Initial states:** $\mu_{s0} = 1$, $\sigma_{s0} = 0.1$, $\mu_{p0} = 1$, $\sigma_{p0} = 0.1$, $\mu_{n1} = 0.0$, $\sigma_{n2} = 0.01$,$\mu_{n2} = 0.0$, $\sigma_{n1} = 0.01$
- **Taste shocks:** $\sigma_\epsilon = 0.1$

B.2 Hyper-parameters

B.3 Value function iteration

The DP-solution follows roughly the same format as the DL-solution with the exception that we do not solve the keeper and adjuster problems simultaneously. The algorithm is a standard backwards induction loop with the following elements in each period:

1. If not last-period: Compute post-decision value function on grids of post-decision states: $\overline{v}(p_t, \overline{m}_t, d_t)$
2. Solve keeper problem:

$$\max_{a_t^0} u(c_t, n_t) + \beta \overline{v}_t(p_t, n_t, \overline{m}_t)$$

$$c_t = m_t(1 - a_t^0)$$

$$\overline{m}_t = a_t^0 m_t$$

	DL-1D	DL-2D
Maximum number of minutes before termination	120.000	180.000
Sample size, N^{train}	150.000	150.000
Number of FOC targets	1.000	1.000
Standard deviation for initialization of weights and biases from normal distribution	0.001	0.001
Number of value networks	3.000	3.000
Neurons in the policy network	[500 500]	[500 500]
Neurons for value network	[500 500]	[500 500]
Number of quadrature points	16	16
Batch size for training	150	150
Size of replay buffer	8	8
Initial exploration noise, σ_ϵ	0.100	0.100
Decay rate for exploration noise	1.000	1.000
Minimum exploration noise	0.000	0.000
Initial learning rate for the policy network	0.001	0.001
Decay rate for policy learning rate	1.000	1.000
Minimum learning rate for the policy network	0.000	0.000
Learning rate maximum for value functions	0.001	0.001
Decay in learning rate for value functions	1.000	1.000
Minimum learning rate for value functions	0.000	0.000
Activation function for policy network final layer	sigmoid	sigmoid
Activation function for policy network intermediate layers	relu	relu
Simulation frequency, Δ_R	10	10
Start training policy after this number of episodes	50	50
Tolerance for policy loss	0.000	0.000
Activation functions for value network intermediate layers	relu	relu

Table B.1: Hyper-parameters

	t =	0		4		9		14		19	
	Model	DP	DL	DP	DL	DP	DL	DP	DL	DP	DL
var	moment										
m	Mean	1.00	1.00	1.16	1.16	1.24	1.24	1.17	1.16	1.01	1.01
	Median	0.99	0.99	1.11	1.11	1.17	1.17	1.09	1.09	0.96	0.96
	Variance	0.01	0.01	0.10	0.10	0.18	0.18	0.18	0.18	0.11	0.11
	Skewness	0.30	0.30	0.86	0.87	1.14	1.13	1.26	1.25	1.01	1.02
	Kurtosis	3.16	3.16	4.15	4.15	5.15	5.09	5.86	5.81	4.76	4.88
c	Mean	0.59	0.58	0.77	0.77	0.83	0.83	0.91	0.91	1.19	1.21
	Median	0.59	0.58	0.76	0.76	0.81	0.81	0.88	0.88	1.13	1.15
	Variance	0.00	0.00	0.02	0.02	0.04	0.04	0.06	0.06	0.14	0.14
	Skewness	0.17	0.19	0.57	0.59	0.70	0.74	0.83	0.83	1.09	0.96
	Kurtosis	3.03	3.11	3.56	3.58	3.76	3.93	4.40	4.33	4.85	4.54
d_1	Mean	0.41	0.42	0.94	0.95	1.18	1.18	1.14	1.14	0.45	0.43
	Median	0.40	0.41	0.92	0.92	1.13	1.14	1.09	1.09	0.42	0.36
	Variance	0.01	0.01	0.08	0.08	0.14	0.14	0.13	0.13	0.05	0.05
	Skewness	0.69	0.45	0.55	0.53	0.70	0.72	0.90	0.89	1.26	1.39
	Kurtosis	3.90	3.26	3.39	3.35	3.56	3.63	4.39	4.26	5.25	5.13

Table B.2: Moments of the Non-Convex Durables Model ($D = 1$)

3. Solve adjuster problem:

$$\begin{aligned}
\max_{a_t^1, a_t^2} u(c_t, d_t) + \beta \overline{v}_t(p_t, n_t, \overline{m}_t) c_t = m_t(1 - a_t^0) \overline{m}_t = a_t^0 m_t \\
x_t = m_t + (1 - \kappa) n_t \\
\overline{m}_t = a_t^1 x_t \\
c_t = (1 - a_t^1) a_t^2 x_t \\
d_t = (1 - a_t^1)(1 - a_t^2) x_t
\end{aligned}$$

A similar algorithm is used for the $D = 2$ case.

t =	0		4		9		14		19	
Model	DP	DL	DP	DL	DP	DL	DP	DL	DP	DL
Corr(m,c)	0.73	0.67	0.75	0.75	0.81	0.81	0.87	0.87	0.90	0.89
Corr(m,d_1)	0.93	0.94	0.47	0.47	0.51	0.51	0.66	0.66	0.64	0.62
Corr(m,p)	0.01	0.01	0.73	0.73	0.75	0.76	0.82	0.82	0.94	0.94
Corr(c,d_1)	0.43	0.38	0.39	0.39	0.62	0.62	0.77	0.77	0.37	0.32
Corr(c,p)	0.64	0.73	0.95	0.95	0.97	0.97	0.97	0.97	0.87	0.87
Corr(d_1,p)	-0.35	-0.33	0.54	0.54	0.68	0.67	0.78	0.78	0.64	0.61

Table B.3: Correlations of the Non-Convex Durables Model ($D = 1$)

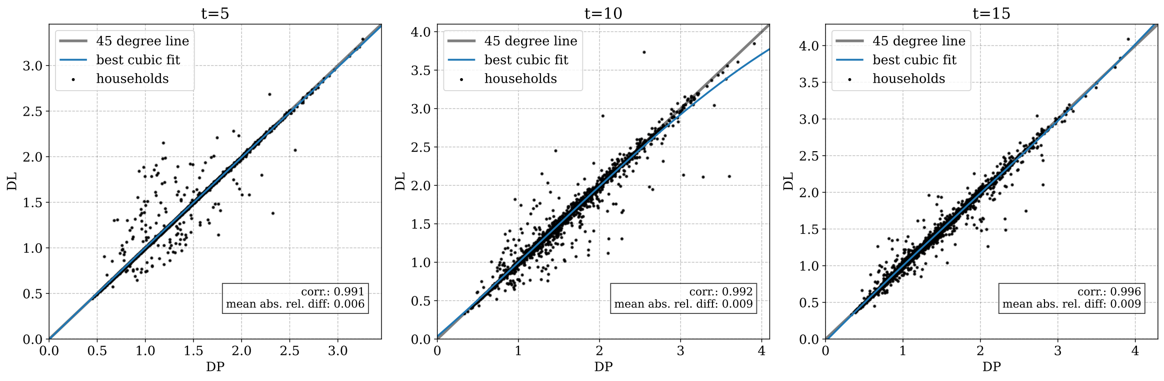

Figure B.2: Cross-Section of cash-on-hands in the DL and DP Solutions for $D = 1$

Notes: Each panel shows the distribution of cash-on-hand in the DP vs DL solution, for a different period. If both solutions perfectly coincided, all the points would lie on the 45° line.

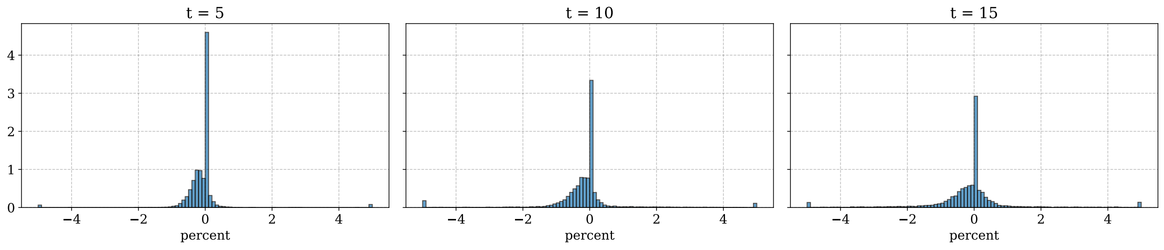

Figure B.3: Relative distances between cash-on-hand across DP and DL

Notes: Each panel plot the share of difference of cash-on-hands $(m^{DL} - m^{DP})/m^{DP}$ between the two solutions, over time.

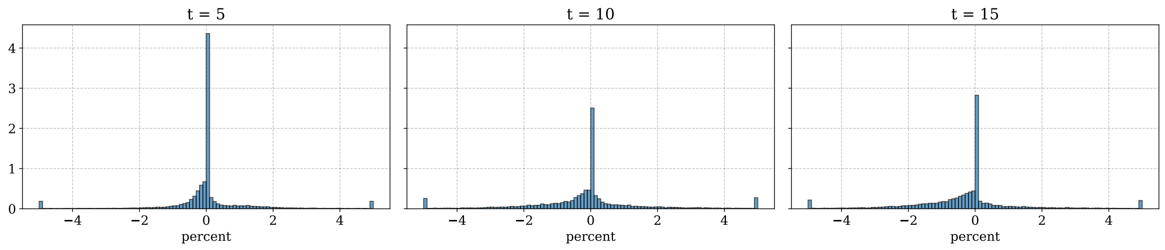

Figure B.4: Relative distances between cash-on-hand across DP and DL with $D = 2$

Notes: Each panel plot the share of difference of cash-on-hands $(m^{DL} - m^{DP})/m^{DP}$ between the two solutions, over time.

B.4 $D = 1$

B.5 $D = 2$

B.6 Warm-starting

In this section, we investigate a »warm-starting« algorithm. The main idea is to take an existing solution (i.e. set of parameters for the value and policy functions) as an initial guess for a different model. We illustrate this by using the solution of our baseline one durable goods model as a starting point for a model with a higher discount factor ($\beta = 0.97$ vs $\beta = 0.965$). Figure B.5 compares the average lifetime reward. We find that the warm-started solution is very accurate even early in training.

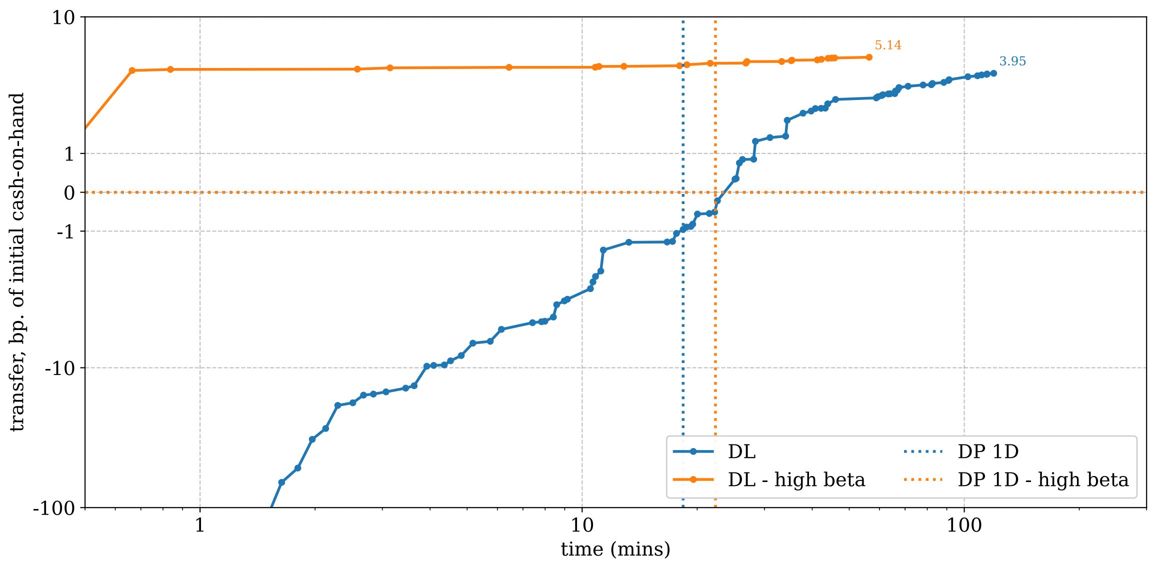

Figure B.5: Lifetime-Reward with Warm-Starting

Notes: The x-axis shows time in minutes, while the y-axis shows the transfer required to make agents indifferent between the DP solution and the current DL solution. When below 0, this means that the agent would have demanded a transfer of x units of cash-on-hand to use the DL solution instead of the DP one. When above 0, it means that the agent would have demanded a transfer of x units of cash-on-hand to use the DP solution instead of the DL one. We only show the transfer when the algorithm improves.

C Large life-cycle model

C.1 Summary

We have the following state variables:

1. Portfolio block:
   (a) a_t: risky asset position
   (b) b_t: safe asset position
2. Job ladder block:
   (a) e_t: employment opportunity
   (b) w_t: current wage
   (c) p_t: stock of human capital
3. Housing block:
   (a) h_t: stock of housing
   (b) L_t: stock of mortgage

The following discrete-choices:

1. Portfolio block: $d_t^p \in \{K, A\}$, where K is being a keeper, A an adjuster
2. Housing block: $d_t^h \in \{R, O, B\}$ where$R$ is being a renter,O an owner,B a buyer
3. Job-ladder block: $d_t^e \in \{U, E\}$where U is not working (unemployed), E is employed.

The following continuous-choices:

1. The savings rate: $s_t^s = \begin{cases} \frac{b_{t+1}}{m_t} & \text{if } d_t^p = K \\ \frac{a_{t+1}+b_{t+1}}{m_t} & \text{if } d_t^p = A \end{cases}$
2. Risky portfolio share: $s_t^p = \begin{cases} 0 & \text{if } d_t^p = K \\ \frac{a_{t+1}}{a_{t+1}+b_{t+1}} & \text{if } d_t^p = A \end{cases}$
3. Housing spending share: $s_t^h = \begin{cases} \frac{\tilde{h}_t r_t^h}{(1-s_t^s)m_t} & \text{if } d_t^h = R \\ 0 & \text{if } d_t^h = O \\ \frac{p_t^h h_{t+1} - L_{t+1}}{(1-s_t^s)m_t} & \text{if } d_t^h = B \end{cases}$

We have the following post-decision state transition functions:

1. Portfolio block

(a) Risky assets: $\bar{a}_t = \begin{cases} a_t & \text{if } d_t^p = K \\ m_t s_t^s s_t^p & \text{if } d_t^p = A \end{cases}$

(b) Safe assets: $\bar{b}_t = \begin{cases} m_t s_t^s & \text{if } d_t^p = K \\ m_t s_t^s (1 - s_t^p) & \text{if } d_t^p = A \end{cases}$

2. Housing block

(a) Housing stock: $h_{t+1} = \bar{h}_t = \begin{cases} 0 & \text{if } d_t^h = R \\ h_t & \text{if } d_t^h = O \\ m_t(1 - s_t^s) s_t^h / (p_t^h (1 - s_t^L \lambda^L)) & \text{if } d_t^h = B \\ h_t & \text{if } d_t^h = R \end{cases}$

(b) Mortgage: $L_{t+1} = \bar{L}_t = \begin{cases} 0 & \text{if } d_t^h = R \\ L_t(1 + r^L) - f_t(L_t) & \text{if } d_t^h = O \\ s_t^L \lambda^L p_t^h \bar{h}_t & \text{if } d_t^h = B \\ s_t^L \lambda^L p_t^h h_t & \text{if } d_t^h = R \end{cases}$

3. Job-ladder block

(a) Discrete labor choice this period: $\bar{\ell}_t = \begin{cases} 0 & \text{if } d_t^e = U \\ 1 & \text{if } d_t^e = E \end{cases}$

(b) Wage this period: $\bar{w}_t = \begin{cases} 0 & \text{if } d_t^e = U \\ w_t & \text{if } d_t^e = E \end{cases}$

(c) Human capital: $\bar{p}_t = \begin{cases} p_t & \text{if } d_t^e = U \\ p_t & \text{if } d_t^e = E \end{cases}$

Also note that housing utility flow is given by

$$\tilde{h}_t = \begin{cases} s_t^h (1 - s_t^s) m_t & \text{if } d_t^h = R \\ h_t & \text{if } d_t^h = O \\ \bar{h}_t & \text{if } d_t^h = B \end{cases}$$

Finally, the state transition functions are given by

1. Portfolio block:

(a) $a_{t+1} = \bar{a}_t(1 + r_t^a)$ where r_t^a is a shock

(b) $b_{t+1} = \bar{b}_t(1 + r^b)$

2. Housing block:
   (a) $h_{t+1} = \bar{h}_t$
   (b) $L_{t+1} = \bar{L}_t$
3. Job-ladder block:
   (a) $e_{t+1} = \begin{cases} \varepsilon^e_{t+1} & \text{if } d^e_t = U \\ 1 - \varepsilon^u_{t+1} & \text{if } d^e_t = E \end{cases}$
   (b) $w_{t+1} = \begin{cases} \varepsilon^e_{t+1}\omega_{t+1} & \text{if } d^e_t = U \\ (1 - \varepsilon^e_{t+1})w_t + \varepsilon^e_{t+1}\omega_{t+1}1_{\omega_{t+1} > w_t} & \text{if } d^e_t = E \end{cases}$
   (c) $\log p_{t+1} = \max\left(\log p_t + \begin{cases} \mu^{p,u} & \text{if } d^e_t = U \\ \mu^{p,e} & \text{if } d^e_t = E \end{cases} + \sigma^p \varepsilon^p_t, \underline{p}\right)$

C.2 Hyper-parameters

	DL
Maximum number of minutes before termination	720.000
Sample size, N^{train}	150.000
Standard deviation for initialization of weights and biases from normal distribution	0.001
Number of value networks	3.000
Neurons in the policy network	[500 500]
Neurons for value network	[500 500]
Number of quadrature points	500
Batch size for training	150
Size of replay buffer	8
Initial exploration noise, σ_ϵ	0.250
Decay rate for exploration noise	1.000
Minimum exploration noise	0.000
Initial learning rate for the policy network	0.000
Decay rate for policy learning rate	1.000
Minimum learning rate for the policy network	0.000
Learning rate maximum for value functions	0.001
Decay in learning rate for value functions	1.000
Minimum learning rate for value functions	0.000
Activation function for policy network final layer	sigmoid
Activation function for policy network intermediate layers	relu
Simulation frequency, Δ_R	10
Start training policy after this number of episodes	-1
Tolerance for policy loss	0.000
Activation functions for value network intermediate layers	relu

Table C.1: Hyper-parameters