The Review of Financial Studies

Machine Learning for Continuous-Time Finance

Victor Duarte
University of Illinois at Urbana-Champaign, United States

Diogo Duarte
Florida International University, United States

Dejanir H. Silva
Purdue University, United States

We develop an algorithm for solving a large class of nonlinear high-dimensional continuous-time models in finance. We approximate value and policy functions using deep learning and show that a combination of automatic differentiation and Ito's lemma allows for the computation of exact expectations, resulting in a negligible computational cost that is independent of the number of state variables. We illustrate the applicability of our method to problems in asset pricing, corporate finance, and portfolio choice and show that the ability to solve high-dimensional problems allows us to derive new economic insights. (*JEL* G11, G12, G32)

Received: October 18, 2018; Editorial decision: February 11, 2024
Editor: Itay Goldstein
Authors have furnished an Internet Appendix, which is available on the Oxford University Press Web site next to the link to the final published paper online.

Dynamic programming is one of the cornerstones of modern financial economics. The behavior of investors, managers, households, and governments is typically represented as the result of maximizing their respective value functions. Dynamic programming is, however, plagued by the "curse of

This paper benefited from comments by Markus Brunnermeier, Julia Fonseca, Daniel Greenwald, Zhengyang Jiang, Leonid Kogan, Deborah Lucas, Karel Mertens, Alexis Montecinos, Jonathan Parker, Alex Richter, Adrien Verdelhan, Gianluca Violante, and seminar participants at the WEAI Annual Meeting, the Macro Financial Modeling Summer Session, the MIT Finance Seminar, Princeton, New York Fed, Dallas Fed, UT Dallas, John Hopkins Carey, Rice Jones, UIUC Gies College of Business, Florida International University, Purdue University. We thank Zibo Zhou for his excellent research assistance. Generous financial support for this project was provided by The Becker-Friedman Institute's Macro Financial Modeling Initiative. We have also benefited from the valuable suggestions of Stijn Van Nieuwerburgh and Itay Goldstein (the Editors) and three anonymous referees. Supplementary data can be found on *The Review of Financial Studies* web site. Send correspondence to Victor Duarte, vduarte@illinois.edu.

The Review of Financial Studies 37 (2024) 3217–3271

https://doi.org/10.1093/rfs/hhae043 Advance Access publication September 4, 2024

dimensionality" (Bellman 1957), that is, it becomes exponentially more challenging in terms of computing time and memory as the number of state variables increases. The curse of dimensionality encompasses three separate challenges, sometimes referred to as the three curses of dimensionality (Powell 2007). The first curse refers to the challenge of approximating a high-dimensional nonlinear function on a computer. The second curse of dimensionality refers to the computation of expectations involved in Bellman equations. Finally, the third curse is maximizing an objective function at each iteration step. Each of these challenges has severely limited the advancement of financial economics. Therefore, most financial research today is restricted to models featuring either small state spaces or linearized solutions.[1]

This paper proposes a novel algorithm that handles nonlinear stochastic dynamic programming problems with large state spaces, addressing the three curses of dimensionality and opening up the possibility of studying models set in a richer economic environment. To address the first curse of dimensionality, we use deep neural networks to represent value functions and optimal policies. To overcome the second curse of dimensionality, we show how to combine the autodifferentiation feature of modern machine-learning libraries and Ito's lemma to efficiently compute exact expectations in continuous-time dynamic systems driven by Brownian shocks.[2] To overcome the third curse of dimensionality, we employ a version of the generalized policy iteration of Sutton and Barto (1998) based on policy gradients (Lillicrap et al. 2015). For this reason, we refer to our method as *deep policy iteration* (DPI hereafter), as it combines value and policy function approximations using deep neural networks and generalized policy iteration to handle high-dimensional problems.

We then apply our method to a range of problems in finance. These applications serve two main purposes. First, they illustrate our method's versatility by showing how to handle different problems involving features such as large state spaces, kinks, and jumps or by showing how to efficiently perform global sensitivity analysis in structural models. Second, they enable us to document the performance and accuracy of our method in the context of standard finance problems, as well as to compare our solution to leading alternative numerical methods, such as the Smolyak-based projection method and finite differences.

Different from previous work that used *shallow* neural networks to solve or estimate economic models (Haugh and Kogan 2004; Norets 2012), we propose using *deep learning* to approximate value and policy functions.[3] Deep learning is fundamentally different from classical machine learning, as

[1] We call a state space *small* if it has fewer than five dimensions and *large* otherwise.

[2] Throughout the paper, the term *exact* should be interpreted as exact up to machine precision.

[3] Shallow networks are neural networks with a single hidden layer. Section 1 defines neural networks and hidden layers.

it requires an entirely new ecosystem of software, hardware, and methods that were only recently developed. Starting with Mnih et al. (2015), deep learning has emerged to become the de-facto technology for functional approximation in *reinforcement learning*, the subfield of machine learning that studies intertemporal optimization, being successfully deployed to solve problems with hundreds of state variables.[4]

In contrast to reinforcement learning applications, we make explicit use of the state dynamics to develop a much more efficient algorithm for the types of problems financial economists study. In a continuous-time setting, we implement an efficient algorithm to compute instantaneous drifts and volatilities for arbitrary functions. We show that the computational cost of evaluating the drift and volatility does not scale with the number of state variables. Furthermore, that cost scales less than linearly with the number of shocks. This allows us to compute exact expectations required to perform Bellman iterations.

Finally, we use policy gradients (Lillicrap et al. 2015) to improve the policy function at each policy iteration step. This approach consists in gradually improving the policy function using only the gradient at each step. Since gradients can be computed with negligible cost by using *backpropagation* (Rumelhart, Hinton, and Williams 1988), this addresses the third curse of dimensionality.

To illustrate the broad applicability of our method, we consider large-dimensional problems in three core areas of finance: asset pricing, corporate finance, and portfolio choice. For asset pricing, we consider the Lucas orchard economy of Martin (2013), a multitree extension of the classical one-tree exchange economy of Lucas (1978). We show that the DPI algorithm is able to solve a Lucas exchange economy with up to 100 trees while sustaining low root-mean-square error (RMSE hereafter). Moreover, we show that the time-to-solution scales approximately linearly with the number of state variables, illustrating our method's ability to alleviate the curse(s) of dimensionality. In contrast, the Smolyak projection method, a numerical method commonly used to handle large-dimensional problems, fails to sustain low RMSEs as the state space grows. More importantly, the Smolyak method quickly exhausts computer memory and is unable to produce a solution for an economy with more than 25 trees.

We also show that the focus on low-dimensional problems, an assumption typically made for tractability reasons, may have important economic implications. In particular, we argue that many of the interesting asset pricing effects found in the cases of two trees (Cochrane, Longstaff, and Santa-Clara 2008), typically involving the behavior of small firms, disappear as we increase the number of trees. The reason is that, with only two trees, either both trees are

[4] See, for instance, Silver et al. (2016, 2017) and Heess et al. (2017).

of similar size, and the economy is well-diversified, or we have one tree that is small, and the economy is severely underdiversified. In contrast, with a large number of trees, it is possible to study small firms in reasonably diversified economies. While changes in the dividend share of a small firm have a large impact on aggregate consumption volatility for underdiversified economies, this is not the case when the economy is more diversified. We show that the strong valuation effects for small firms found with just a few trees disappear as we increase the number of trees, as their impact on aggregate volatility becomes more muted. Therefore, the ability to solve high-dimensional problems may allow us to relax assumptions made based only on tractability and instead focus on the assumptions that are of economic interest.

Our second application is a dynamic corporate finance model, in the spirit of Hennessy and Whited (2007), where firms face equity issuance and investment adjustment costs. An important feature of this application is that the solution may feature kinks, as the marginal incentives to invest vary depending on whether the firm is issuing equity or paying dividends (or neither). To show the method's ability to solve this problem, we compare our solution to the one from a finite differences method with a fine grid, which we use as our benchmark. Our findings closely match the results from finite differences, indicating the accuracy of our solution. Therefore, our method can handle problems with severe nonlinearities, even when a classical solution to the continuous-time problem is not available, such as in the case of the problems with kinks.[5]

For any given value of the parameters, our version of the Hennessy-Whited model can be solved using standard methods, such as finite differences. However, we are often interested in the solution for a very large number of parameter values. For instance, to be able to show which features of the data are particularly informative about a given parameter, one needs to show how equilibrium moments change with the parameters, which can be computationally very costly. We show how to perform global sensitivity analysis in an efficient manner by including as inputs of the network not only the state variables but also the parameters of interest.[6] In our application, this requires effectively solving a problem with seven state variables, the two original states plus five parameters. As a result, we obtain the model's solution for any point of the state space or the parameter space. By simultaneously solving for an entire class of models, our method eliminates the need to repeatedly solve the model for each new parameter value, which gives an efficient way of assessing how parameters affect the model predictions. This feature is potentially useful when performing structural estimation.[7]

[5] For a recent discussion of viscosity solutions, the appropriate solution concept when the value function is not differentiable everywhere (see, e.g., Achdou et al. 2022).

[6] On the importance of sensitivity analysis for structural work, see, for example, Andrews, Gentzkow, and Shapiro (2017) and Catherine et al. (2022).

[7] For an application of these ideas to the context of structural estimation, see Duarte (2018).

In our third application, we show how the DPI algorithm can be used to solve a portfolio choice problem in which the interest rate and risk premium are time-varying and driven by a large number of return predictors. Since closed-form solutions are typically not available for high-dimensional portfolio problems, we propose a new way to assess the accuracy of our method. In particular, we reverse engineer the process for the interest rate and the risk premium such that the policy functions are any given closed-form expressions. We can then solve the portfolio problem with the reverse-engineered process for the returns using the DPI method and then compare our solution to the known closed-form expressions. This process of reverse engineering a problem provides an effective laboratory for evaluating the performance of our solution method for high-dimensional problems. We find that the DPI method provides accurate solutions even with 10 return predictors and captures a wide range of relationships between the portfolio share and a return predictor, depending on the region of the state space.

Having demonstrated DPI's ability to solve high-dimensional nonlinear portfolio choice problems, we proceed to analyze optimal asset allocation in an empirically motivated model with multiple risky assets and realistic return dynamics. The optimal portfolio features a substantial degree of market timing. At times, the investor is heavily invested in stocks, such as in the early 1950s and 1960s, and sometimes the investor is nearly out of the stock market, as in the early 1970s or early 2000s. Moreover, macroeconomic variables, and in particular fiscal variables, explain a sizeable fraction of the variation in portfolio shares.

To keep the exposition as simple as possible, we focus on the case of Brownian shocks and economies with a representative agent for our three applications. However, with minor modifications, our methods can also be applied to models with jumps. In Appendix B, we solve the model of time-varying disasters in Wachter (2013). One important distinction relative to models with Brownian shocks is that expectations appear explicitly even in continuous time. We show that, by using simulation methods analogous to the least-squares Monte Carlo method of Longstaff and Schwartz (2001), we can apply the DPI algorithm even in problems with jumps. We compare our solution to the closed-form expression provided by Wachter (2013) and show that our method accurately captures the behavior of an economy subject to rare disasters.

Our work is related to the rapidly growing literature on machine-learning applications in finance. In recent years, we have witnessed rapid adoption of these techniques in several domains of finance, such as asset pricing (Gu, Kelly, and Xiu 2020; Bianchi, Büchner, and Tamoni 2021; Chen, Pelger, and Zhu 2023), corporate finance (Li et al. 2021; Cao et al. 2023), derivatives and credit markets (Duarte et al. 2020; Chen, Didisheim, and Scheidegger 2021; Sadhwani, Giesecke, and Sirignano 2021; Fuster et al. 2022; Bali et al. 2023), among others. These applications focus on the use of machine-learning

techniques for reduced-form empirical work, while our focus is on numerical methods for structural models.[8]

Our paper is also related to the literature using finite-difference methods (Achdou et al. 2022; Brunnermeier and Sannikov 2014; Ahn et al. 2018) or projection methods (Moreira and Savov 2017; Drechsler, Savov, and Schnabl 2018; Kargar 2021) in continuous time. While these methods are only suitable for small-scale problems, we show how to use deep learning, combined with an efficient way to compute Hamilton-Jacobi-Bellman equations with Brownian shocks, to handle large-scale problems.[9]

Since this paper was first made publicly available, a number of articles have employed related methods and have adopted deep learning for solving or estimating nonlinear dynamic problems in economics. Applications include structural estimation (Duarte 2018; Chen, Didisheim, and Scheidegger 2021; Kase, Melosi, and Rottner 2022), models with discrete choice (Maliar and Maliar 2022), business cycles (Bybee et al. 2021; Bretscher, Fernández-Villaverde, and Scheidegger 2022), heterogeneity and wealth distribution (Maliar, Maliar, and Winant 2021; Han, Yang et al. 2021; Azinovic, Gaegauf, and Scheidegger 2022; Fernández-Villaverde, Hurtado, and Nuno 2023), life cycle models (Duarte et al. 2021), macro-finance models (Gopalakrishna 2021; Sauzet 2021), and climate economics and finance (Folini et al. 2024), among others. Despite recent rapid advancements in the field, our approach stands out. We have innovatively combined a gradient-based generalized policy iteration method, which eliminates the need for root-finding routines, with a cost-effective computation of the value function drift, to effectively address the three curses of dimensionality. Our approach will enable researchers to delve into high-dimensional problems in financial economics.

1. Machine Learning

This section covers basic machine-learning concepts and methods needed to implement the algorithm presented in Section 2. For excellent textbook treatments, see Sutton and Barto (1998) and Goodfellow, Bengio, and Courville (2016). The reader who is already familiar with deep learning and generalized policy iteration may want to skip to the next section.

1.1 Supervised learning and neural networks

The goal of supervised learning is, broadly speaking, to learn how to represent functions. For a concrete example, consider a set of observations $\{X_i, Y_i\}_{i=1}^N$ and suppose that we are interested in constructing a function V such that

[8] For a recent discussion of these applications in asset pricing, with a focus on shrinkage methods, see, for example, Nagel (2021).

[9] For an early use of machine-learning techniques in discrete time, see the work by Scheidegger and Bilionis (2019) on Gaussian processes.

$V(X_i) = Y_i$. For instance, X_i may be a digital picture and Y_i an indicator of whether a particular person appears in the image. Since a grayscale digital picture with one megapixel, for example, has one million dimensions, listing all possible combinations of X_i and Y_i in a lookup table is an impossibly difficult task. The machine-learning solution for this problem is to assume a flexible parametric function $V(X_i; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector of parameters, and use data to recover $\boldsymbol{\theta}$. To represent this highly nonlinear function, we need functional forms that can capture complex and nonlinear interactions between the regressors. A particularly powerful set of function approximators is the class of neural networks.

The starting point of constructing a neural network is building a linear model of the type $Y_i = \langle \mathbf{W}_0, \mathbf{X}_i \rangle + b_0$, where $Y_i \in \mathbb{R}$ is the dependent variable, $\mathbf{X}_i \in \mathbb{R}^d$ is a data point, $\mathbf{W}_0 \in \mathbb{R}^d$ is a vector of coefficients, $b_0 \in \mathbb{R}$ is a coefficient (i.e., bias), and the operator $\langle \cdot, \cdot \rangle$ represents the inner product in $\mathbb{R}^d$. The next step is to apply a nonlinear function $\sigma(\cdot)$, known in the literature as an activation function, to the output. Figure 1 shows three commonly used activation

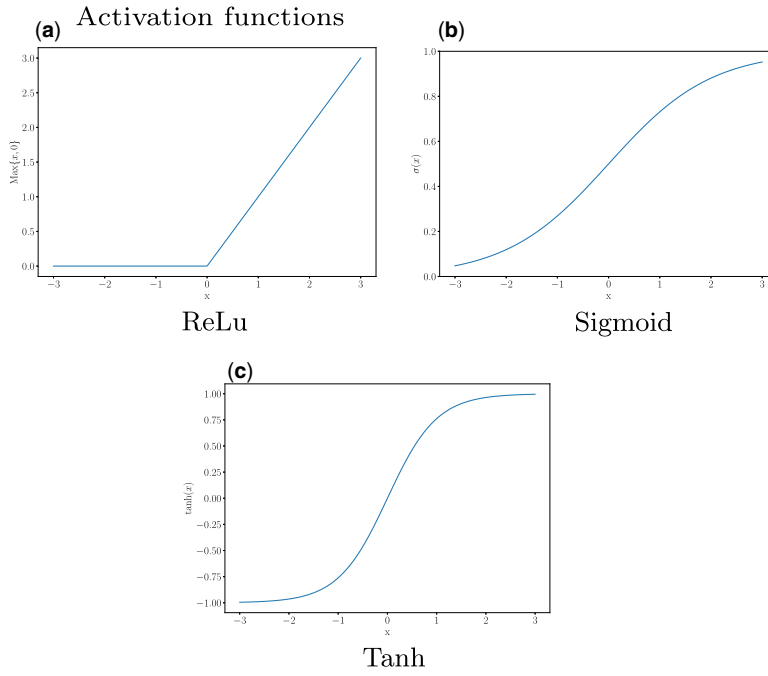

Figure 1
Activation functions
Panel A shows the rectified linear unit (ReLu), the most common activation function used in machine learning applications. Panels B and C show two possible alternatives, the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ and the hyperbolic tangent $\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$.

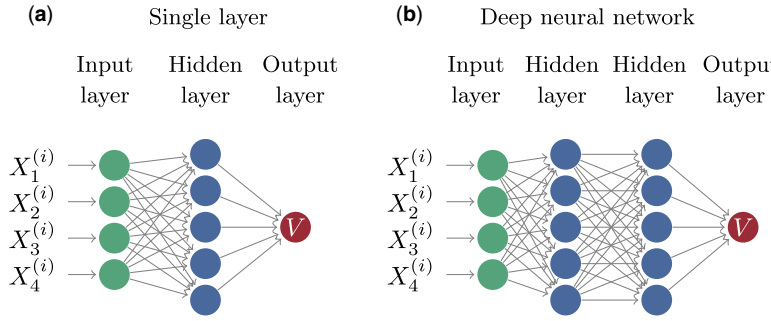

Figure 2
Feedforward neural network

The green circles represent each entry of the input vector $\mathbf{X}^{(i)} = \left[X_1^{(i)}, X_2^{(i)}, X_3^{(i)}, X_4^{(i)}\right]^\top$. The hidden units are represented by blue circles. Each hidden unit performs a composition of a nonlinear function (activation function) and a linear transformation of the outputs of the previous layer. The outputs of the final hidden layer are combined linearly to produce the final output. $\boldsymbol{\theta}$ is the collection of all parameters of the network.

functions. Panel A shows the rectified linear unit (Jarrett, Kavukcuoglu, and LeCun 2009), which is the default choice in most applications, while panels B and C show the sigmoid and hyperbolic tangent activation functions.

Let $\mathbf{G}_0 = \sigma(\langle \mathbf{W}_0, \mathbf{X}_i \rangle + b_0)$ be the output of this nonlinear function, known in the literature as the hidden unit. When we perform this operation on a set of vectors and coefficients $\left\{\mathbf{W}_j, b_j\right\}_{j \in 0,1,\ldots,n_{G-1}}$ and stack them into a vector $\mathbf{G} \in \mathbb{R}^{n_G}$, we obtain a hidden layer. In the final step, a single-layer neural network takes a linear combination of $\mathbf{G}$ to produce the final output $Y = \langle \mathbf{G}, \mathbf{W}_{n_G} \rangle + b_{n_G}$.

Panel A of Figure 2 shows a neural network with five hidden units. When this neural network is extended by adding many hidden layers, stacked on top of each other, it receives the name of a deep neural network. Panel B of Figure 2 shows a deep neural network with two hidden layers. The number of hidden layers, also known as the depth of the neural network, is an important feature to accurately capture nonlinear relationships. Empirically, deep neural networks have been found to perform much better than single-layer networks.[10]

An important theoretical result in the neural network literature is the so-called "Universal Approximation Theorem," which states that any continuous function on compact subsets of $\mathbb{R}^n$ can be uniformly approximated by enough hidden units (Cybenko 1989; Hornik 1991).[11] This result may be familiar to financial economists who know the Options Spanning Theorem of Ross (1976) that states that any contract can be formed as a portfolio of options. Indeed, in the particular case where (i) the activation function $\sigma(\cdot)$ is the rectified linear

[10] See chapter 6 of Goodfellow, Bengio, and Courville (2016) and the references therein.

[11] To state this more precisely, this means the theorem shows that the set of linear combinations of sigmoidal activation functions is dense in the set of continuous functions on the unit cube.

unit, (ii) the input X is scalar, and (iii) the weights are unit weights ($W_j = 1\ \forall j$), the output of the jth hidden unit is the payoff of a call option on X with strike $-b_j$. Thus, the output layer combines many call options to produce a given payoff. In our options analogy, a two-layer neural network would correspond to a portfolio of options on portfolios of call options.

If the output of a neural network has to satisfy some model-implied constraints, we can apply a final nonlinear transformation to ensure that the constraints are not violated. For instance, if a network represents consumption choice, we can apply the exponential or softplus functions as the final transformation to impose nonnegativity. Likewise, sigmoid or hyperbolic tangent functions can be used to bound functions.

1.2 Stochastic gradient descent and backpropagation

The standard (nonstochastic) method of gradient descent (or simply, steepest descent) of Cauchy (1847) consists of moving the parameter $\boldsymbol{\theta}$ of the parametric representation of $V(\mathbf{X})$, represented by $V(\mathbf{X};\boldsymbol{\theta})$, in the direction that minimizes some measure of error the fastest. A natural measure of fitness is the one-half mean-squared error (MSE hereafter) over N observations, that is,

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}(V(\mathbf{X}_i;\boldsymbol{\theta}) - Y_i)^2.$$

Starting with an initial guess $\boldsymbol{\theta}$, the gradient descent algorithm updates $\boldsymbol{\theta}$ according to

$$\begin{aligned}\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})\\ &= \boldsymbol{\theta} - \eta\frac{1}{2N}\sum_{i=1}^{N}\nabla_{\boldsymbol{\theta}}(V(\mathbf{X}_i;\boldsymbol{\theta}) - Y_i)^2,\end{aligned} \tag{1}$$

where η is the learning rate and $\nabla_{\boldsymbol{\theta}}$ denotes the gradient operator with respect to the parameter vector $\boldsymbol{\theta}$.

The key insight of the Stochastic Gradient Descent (SGD hereafter) algorithm is to approximate the expectation (i.e., average) in Equation (1) with a small independent and identically distributed (i.i.d.) sample of the data set $\{\mathbf{X}_i, Y_i\}$. Thus, for $n \ll N$, we can approximate Equation (1) by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\frac{1}{n}\sum_{i\in I_n}(V(\mathbf{X}_i;\boldsymbol{\theta}) - Y_i)\nabla_{\boldsymbol{\theta}}V(\mathbf{X}_i;\boldsymbol{\theta}), \tag{2}$$

where I_n is a random i.i.d. sample of $\{1,2,\ldots,N\}$ with n points.[12] This subsample of points used to approximate the gradient is called the mini-batch.

[12] Typical values for n and N are 128 and 1,000,000 (see, e.g., Krizhevsky, Sutskever, and Hinton 2012). For guidelines on how to choose the batch size, see Goodfellow, Bengio, and Courville (2016).

The use of stochastic methods to compute the MSE loss is one of the key aspects that separate machine learning from pure optimization, and it is essential to make machine learning feasible in high-dimensional problems. As Goodfellow, Bengio, and Courville (2016) explain, computing the MSE loss for a sample with 10,000 observations is 100 times more costly in terms of computational resources than performing the same computation for a sample with 100 observations but only reduces the standard deviation of the gradient of the larger sample by a factor of 10 since the standard error of the mean scales with the square root of the number of observations.

A critical aspect of the iteration in Equation (2) is that it involves all first-order partial derivatives of the network V with respect to its parameters. Therefore, a naive finite-difference approach to compute the derivatives would be too costly. For example, if the network has 100,000 parameters, we would need to compute $V(\mathbf{X}_i;\boldsymbol{\theta}+\varepsilon\mathbf{e}_j)$ for every $j\in\{1,\ldots,100,000\}$, with $\varepsilon\in\mathbb{R}$, and $\mathbf{e}_j\in\mathbb{R}^{100,000}$ is the canonical basis vector in the jth direction. Fortunately, machine-learning software relies on a more efficient method of computing partial derivatives, called backpropagation (Rumelhart, Hinton, and Williams 1988). This algorithm is based on the sequential application of the chain rule, starting from the final layer and moving backward to the initial layer. It can be shown that computing all first-order partial derivatives using backpropagation always has the same cost as computing the function itself.[13] Compared to a finite-difference approach applied to the example above, backpropagation provides an economy of five orders of magnitude.

1.3 Discrete-time Markov decision process

Throughout the paper, we assume that infinitely lived agents face a Markov decision process; that is, there exists a vector of states $\mathbf{s}\in\mathcal{S}\subset\mathbb{R}^n$ that subsumes all relevant information for decision-making. At each instant t, subject to possible environment constraints, the agent chooses a control $\mathbf{c}_t\in\mathcal{A}$ from which she derives instantaneous utility $u(\mathbf{c}_t)$. Her goal is to choose a sequence of controls to maximize the expected value of the sum of discounted future utilities:

$$V^*(\mathbf{s})=\max_{\{\mathbf{c}_t\}_{t=0}^{\infty}}\mathbb{E}\left[\sum_{t=0}^{\infty}\beta^t u(\mathbf{c}_t)|\mathbf{s}\right].$$

The function V^* is called the *optimal state-value function*. The function that maps states to optimal controls $\boldsymbol{\pi}^*:\mathcal{S}\to\mathcal{A}$ is called the *optimal policy function*. More generally, given an arbitrary policy function $\boldsymbol{\pi}:\mathcal{S}\to\mathcal{A}$ (not necessarily optimal), we define the *state-value function associated with* $\boldsymbol{\pi}$ as

[13] See Baydin, Pearlmutter, and Radul (2015) for a survey on backpropagation and other automatic differentiation methods.

$$V_{\boldsymbol{\pi}}(\mathbf{s}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t u(\boldsymbol{\pi}(\mathbf{s}_t))\right].$$

This function represents the expected value of the sum of discounted future utilities for an agent that chooses her controls following the policy $\boldsymbol{\pi}$.

A canonical class of algorithms for solving this Markov decision process is called *policy iteration* (Howard 1960). It consists of iterating between two steps: policy evaluation and policy improvement. As discussed below, a particular case of policy iteration is the canonical *value function iteration* method.

Under technical conditions, the value function $V_{\boldsymbol{\pi}}$ satisfies the Bellman equation

$$V_{\boldsymbol{\pi}}(\mathbf{s}) = u(\boldsymbol{\pi}(\mathbf{s})) + \beta \mathbb{E}\left[V_{\boldsymbol{\pi}}\left(\mathbf{s}'\right)|\mathbf{s}\right], \tag{3}$$

where $\mathbf{s}'$ denotes the state vector next period.[14] The right-hand side of Equation (3) is the Bellman target, and we write it as $T V_{\pi}(\mathbf{s})$.

1.3.1 Direct policy evaluation This functional equation can be solved exactly on a computer only if the state space $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ is finite and the number of states is sufficiently small. In this case, the Bellman equation is linear and can be solved with standard linear algebra tools:

$$\mathbf{V}_{\boldsymbol{\pi}} = (\mathbf{I} - \beta \mathbf{P}_{\boldsymbol{\pi}})^{-1} \mathbf{U}_{\boldsymbol{\pi}},$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{P}_{\boldsymbol{\pi}}$ is the transition probability matrix describing the state dynamics when the agent chooses her controls using the policy $\boldsymbol{\pi}$, $\mathbf{V}_{\boldsymbol{\pi}}$ is the vector of stacked values for every state, $\mathbf{V}_{\boldsymbol{\pi}} = [V_{\boldsymbol{\pi}}(s_1), V_{\boldsymbol{\pi}}(s_2), \ldots, V_{\boldsymbol{\pi}}(s_N)]$, and $U_{\boldsymbol{\pi}}$ is the vector of stacked utilities for every state: $\mathbf{U}_{\boldsymbol{\pi}} = [u(\boldsymbol{\pi}(s_1)), u(\boldsymbol{\pi}(s_2)), \ldots, u(\boldsymbol{\pi}(s_N))]$.

1.3.2 Iterative policy evaluation An alternative algorithm for computing $V_{\boldsymbol{\pi}}$ consists of turning the Bellman equation in Equation (3) into assignments. Starting from an initial arbitrary guess $V_{\boldsymbol{\pi}}^0$, construct a sequence $\left\{V_{\boldsymbol{\pi}}^k\right\}_{k \in \mathbb{N}}$ according to

$$V_{\boldsymbol{\pi}}^k(\mathbf{s}) = T V_{\boldsymbol{\pi}}^{k-1}(\mathbf{s}). \tag{4}$$

This iteration produces the unique solution of Equation (3).

1.3.3 Policy improvement Knowing the value function $V_{\boldsymbol{\pi}}$ associated with the policy $\boldsymbol{\pi}$ makes it possible to find a better policy $\boldsymbol{\pi}' : S \to \mathcal{A}$. Let

$$\boldsymbol{\pi}'(\mathbf{s}) \doteq \arg\max_{\mathbf{c}} \left\{u(\mathbf{c}) + \beta \mathbb{E}\left[V_{\boldsymbol{\pi}}(\mathbf{s}')|\mathbf{s}, \mathbf{c}\right]\right\}. \tag{5}$$

The Policy Improvement Theorem (Bellman 1957; Howard 1960) guarantees that $V_{\boldsymbol{\pi}'}(\mathbf{s}) \geq V_{\boldsymbol{\pi}}(\mathbf{s})$, $\forall \mathbf{s} \in \mathcal{S}$. This step is therefore called *policy improvement*.

[14] For details on the Bellman equation, see Stokey, Lucas, and Prescott (1989) and Ljungqvist and Sargent (2000).

Alternating between policy evaluation and policy improvement is guaranteed to produce the optimal state-value function V^* and the optimal policy $\boldsymbol{\pi}^*$. If the policy evaluation step consists of a single iteration of iterative policy evaluation in Equation (4), the algorithm is called value function iteration.

1.3.4 Large state and action spaces When the number of states is large or takes on a continuum of values, all numerical solution methods have to rely on an approximate version of Equation (3). Likewise, when the action space $\mathcal{A}$ is large, in general, the maximization on the right-hand side of Equation (5) cannot be performed exactly. An algorithm that alternates some approximate version of policy evaluation with an approximate version of policy improvement is called *generalized policy iteration* (Sutton and Barto 1998).

2. Solution Method

In this section, we show how to combine the tools presented in Section 1 with Ito's lemma to solve high-dimensional nonlinear dynamic stochastic problems in continuous time. This combination allows us to efficiently compute exact expectations when the underlying shocks follow Brownian motions, yielding the new and surprising result that the associated computational cost does not increase with the number of states, and increases at most linearly with the number of shocks, therefore avoiding the second curse of dimensionality (Powell 2007).

2.1 Ito's lemma and automatic differentiation

The computational advantage of continuous time over discrete time counterparts is that, in continuous time, expectations can be computed with partial derivatives when the underlying shocks follow Brownian motions. For example, Achdou et al. (2014) and Brunnermeier and Sannikov (2016) present algorithms that perform orders of magnitude faster than their discrete-time counterparts for small-scale problems with one or two state variables. When the state space is low dimensional, one can discretize the state space and approximate partial derivatives with finite differences, and thus computing expectations using Ito's lemma is computationally cheap.

This approach, however, does not scale to problems with a large number of state variables.[15] Consider the vector of state variables $\mathbf{s}$ that follows the stochastic differential equation:

$$d\mathbf{s}_t = \mathbf{f}(\mathbf{s}_t)dt + \mathbf{g}(\mathbf{s}_t)d\mathbf{B}_t, \tag{6}$$

where $\mathbf{s} \in \mathbb{R}^n$, $\mathbf{f}: \mathbb{R}^n \to \mathbb{R}^n$ is the drift, and $\mathbf{g}: \mathbb{R}^n \to \mathbb{R}^{n \times m}$ represents the matrix of loadings on the m-dimensional vector of standard Brownian motions $d\mathbf{B}$.

[15] For example, a 10-dimensional grid with 100 points in each direction requires 10^{17} terabytes of RAM.

Let $V(\mathbf{s})$ denote an arbitrary function of $\mathbf{s}$ with continuous second-order partial derivatives. Ito's lemma states that:

$$\frac{\mathbb{E}dV}{dt}(\mathbf{s}) = \nabla_{\mathbf{s}} V(\mathbf{s})^{\top} \mathbf{f}(\mathbf{s}) + \frac{1}{2} \mathrm{Tr}\left[\mathbf{g}(\mathbf{s})^{\top} \mathbf{H}_{\mathbf{s}} V(\mathbf{s}) \mathbf{g}(\mathbf{s})\right], \tag{7}$$

where $\nabla_{\mathbf{s}} V$ is the gradient and $\mathbf{H}_{\mathbf{s}} V$ the Hessian matrix.

A naive implementation of Ito's lemma would involve computing all first- and second-order partial derivatives, which naturally scales poorly with the number of state variables. The next proposition shows how to bypass these costly computations and avoid the second curse of dimensionality. This result is also part of what distinguishes our algorithm from standard reinforcement-learning implementations, as the state dynamics are typically not known in these applications.

Proposition 1. For a given $\mathbf{s}$, define the auxiliary function $F : \mathbb{R} \to \mathbb{R}$ as

$$F(\epsilon) \equiv \sum_{i=1}^{m} V\left(\mathbf{s} + \frac{\epsilon}{\sqrt{2}} \mathbf{g}_i(\mathbf{s}) + \frac{\epsilon^2}{2m} \mathbf{f}(\mathbf{s})\right), \tag{8}$$

where $\mathbf{g}_i(\mathbf{s})$ represents column i of the matrix $\mathbf{g}(\mathbf{s})$. Then,

$$F''(0) = \frac{\mathbb{E}dV}{dt}(\mathbf{s}). \tag{9}$$

Proposition 1 contains two main insights. First, Equation (9) shows that we can bypass the computation of a multidimensional Ito's lemma on the right-hand side by computing the second derivative of a univariate function on the left-hand side instead. Note that the second derivative of F is effectively a directional derivative of V.[16] Second, since the cost of evaluating a second-order derivative with either backward or forward automatic differentiation is a small multiple of the cost of evaluating $F(0) = V(\mathbf{s})$, the total computational cost of evaluating $\frac{\mathbb{E}dV}{dt}(\mathbf{s})$ is a small multiple of $m \cdot cost(V)$.[17]

To understand the computational gains generated by Proposition 1, consider the following illustrative example where we compute the derivative in Equation (9) using second-order forward mode automatic differentiation. Suppose we have 100 state variables $\mathbf{s}_t = (s_{1,t}, s_{2,t}, \ldots, s_{100,t})$, where each

[16] Formally, $\frac{\mathbb{E}dV}{dt}(\mathbf{s})$ is the sum of the first-order directional derivative $\nabla_{\mathbf{s}} V(\mathbf{s})^{\top} \mathbf{f}(\mathbf{s})$ and the second-order directional derivative $\frac{1}{2} \mathrm{Tr}\left[\mathbf{g}(\mathbf{s})^{\top} \mathbf{H}_{\mathbf{s}} V(\mathbf{s}) \mathbf{g}(\mathbf{s})\right]$.

[17] With forward-mode automatic differentiation, this cost is independent of the number of outputs, while with backward mode it is independent of the number of inputs (for formal bounds, see Griewank and Walther 2008). Since the auxiliary function F has one input and one output, the choice of backward or forward mode is typically not important when using efficient automatic differentiation systems. However, depending on the software, the backward mode can be much slower. Therefore, systematic experimentation is advised to determine the optimal combination of forward and backward modes for superior performance.

component $s_{i,t}$, $i=1,\ldots,100$ has a drift process $\mu_{i,t}$ and a volatility process $\sigma_{i,t}$ on the same Brownian shock dB_t.

Now consider evaluating the function $V(\mathbf{s}_t)=\sum_{i=1}^{100} s_{i,t}^2$ numerically. Squaring each term and adding them all up requires a total of 199 floating point operations (FLOPs), corresponding to 100 multiplications and 99 additions. But if we are interested in computing the drift of V, how many operations do we need to perform when using Proposition 1?

To obtain the drift of V using Proposition 1, we must compute the second derivative of $F(\epsilon)\equiv V\left(\mathbf{s}_t+\epsilon\cdot\frac{\boldsymbol{\sigma}_t}{\sqrt{2}}+\frac{\epsilon^2}{2}\cdot\boldsymbol{\mu}_t\right)$, where $\boldsymbol{\mu}_t=(\mu_{1,t},\mu_{2,t},\ldots,\mu_{100,t})$ and $\boldsymbol{\sigma}_t=(\sigma_{1,t},\sigma_{2,t},\ldots,\sigma_{100,t})$. For a given Taylor series $x=x_0+\epsilon\cdot x_1+\frac{\epsilon^2}{2}\cdot x_2$, automatic differentiation in forward mode produces the Taylor series of a function $f(x)$ by chaining the Taylor series of each elementary function that composes $f(x)$. The function V in this example contains two elementary operations: the square function and the addition, so we only need propagation rules for these two functions. The Taylor expansion of the addition is immediate: the series of the sum is the sum of the series. For the square function, its second-order Taylor expansion yields

$$f(x)=x^2=\left(x_0+\epsilon\cdot x_1+\frac{\epsilon^2}{2}\cdot x_2\right)^2$$

$$=y_0+\epsilon\cdot y_1+\frac{\epsilon^2}{2}\cdot y_2,$$

where $y_0=x_0^2$, $y_1=2\cdot x_0\cdot x_1$, and $y_2=2\cdot(x_0\cdot x_2+x_1^2)$. Note in particular that we need 4 FLOPs to compute the second-order Taylor coefficient y_2: one for the multiplication $x_0\cdot x_1$, one for the multiplication $x_1\cdot x_1$, one for the addition $x_0\cdot x_1+x_1^2$, and one for the multiplication $2\cdot(x_0\cdot x_1+x_1^2)$.

Now consider the original series $x=s_i+\epsilon\cdot\frac{\sigma_i}{\sqrt{2}}+\frac{\epsilon^2}{2}\cdot\mu_i$. In this case, $x_0=s_i, x_1=\frac{\sigma_i}{\sqrt{2}}$, and $x_2=\mu_i$. First, we need 100 FLOPs for the terms $\frac{\sigma_i}{\sqrt{2}}$. To compute the second-order term of the Taylor expansion of the quadratic function, we need another 400 FLOPs, as shown above. Finally, for the summation operation, we need another 99 additions to obtain the second-order derivative of the auxiliary function F. In summary, we need a total of 599 FLOPs to compute the drift of V, a small multiple of the cost of evaluating V itself.

The cost of computing the drift of a high-dimensional function is significantly higher using leading alternative methods. Table 1 shows the computational cost and memory requirements to compute the drift of V using different approaches. As shown, using finite differences to compute all first- and second-order partial derivatives of V to obtain its drift as in Equation (7),

Table 1
Computational cost of numerical derivatives

Method	FLOPs	Memory	Error
1. Finite differences	9,190,800	112,442,048	1.58%
2. Naive autodiff	2,100,501	25,673,640	0.00%
3. Analytical	20,501	44,428	0.00%
4. Proposition 1	599	6,044	0.00%

The table shows the computational cost for computing the drift of $V(s)=\sum_{i=1}^{100} s_i^2$, assuming $s_i=\mu_i=\sigma_i=1$ for $i=1,\ldots,100$, using four different methods: 1) finite differences (with $h=0.001$), 2) a naive use of automatic differentiation (where the Hessian is computed by nested calls to the Jacobian function), 3) using the analytical partial derivatives, and 4) the method described in Proposition 1 combined with forward-mode automatic differentiation. The column FLOPs shows the number of floating point operations required by each approach. The column Memory is measured as bytes accessed. The column Error measures the absolute value of the relative error of each method in percentage terms.

requires over 9 million FLOPs, with a total memory cost of over 112 million bytes. This substantial amount of memory is orders of magnitude larger than the memory usage for the method proposed in Proposition 1, which is about 6,000 bytes.

We emphasize that the large performance difference between the two methods is due to more than the use of automatic differentiation. As shown in Table 1, a naive use of automatic differentiation, where the Hessian is computed by nested calls of the Jacobian function, is only slightly more efficient than finite differences. The reason is that the number of first- and second-order partial derivatives grows rapidly with the number of state variables. By effectively computing a directional derivative as in Proposition 1, we bypass the computation of all these partial derivatives, resulting in this large performance difference. Interestingly, the method proposed in Proposition 1 is more efficient even when the partial derivatives can be computed and evaluated in closed form. As shown in Table 1, 20,501 FLOPs and 44,428 bytes were needed to compute the drift of V using the analytical expressions for the partial derivatives in Equation (7).

The efficiency gains provided by Proposition 1 generalize to more complex functional forms for V. To see how this theoretical result translates into real-world applications, we perform two experiments. In the experiments, we use a more complex functional form than the quadratic function used in the previous illustrative numerical example, and we set V as a two-layer neural network. Panel A shows the cost of computing $\frac{\mathbb{E}dV}{dt}(\mathbf{s})$ as we vary the number of state variables, holding the number of Brownian shocks fixed and equal to one ($m=1$). This cost is defined as the execution time of $\frac{\mathbb{E}dV}{dt}(\mathbf{s})$ divided by the execution time of $V(\mathbf{s})$. As shown, this cost is slightly greater than one, regardless of the number of state variables, showing that evaluating the value-function drift in Equation (9) is essentially as costly as doing a single evaluation of $V(\mathbf{s})$.

Panel B of Figure 3 shows the cost of computing the value-function drift as we vary the number of Brownian shocks, holding fixed the number of state variables at 100. As the number of Brownian shocks increases, the

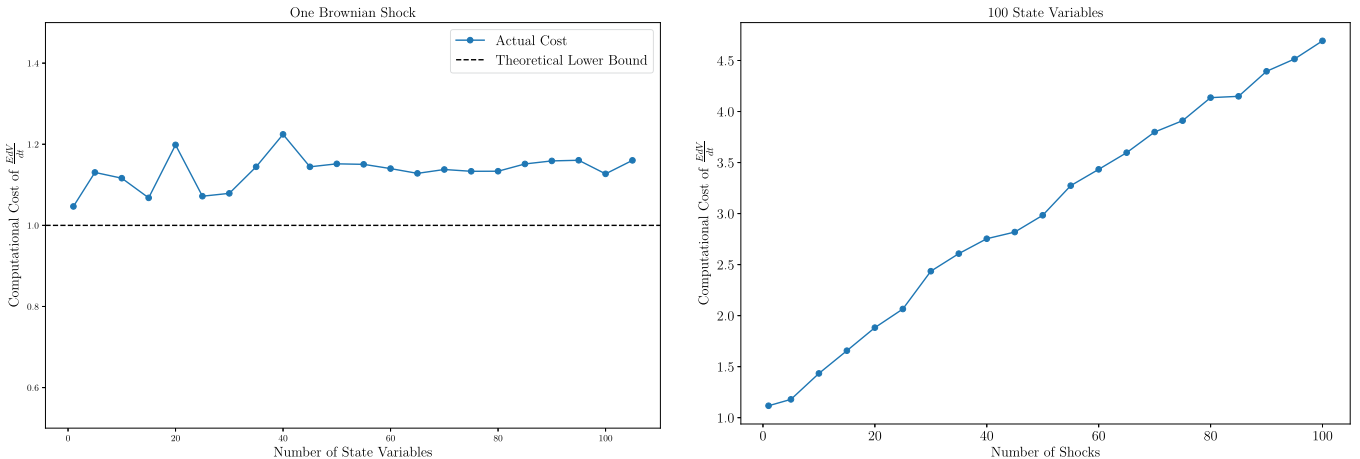

Figure 3
Ito's lemma computational cost
This figure shows how the cost of computing the drift of a function V scales with the number of state variables and with the number of Brownian shocks. We define the cost as the execution time of $\frac{\mathbb{E}dV}{dt}(\mathbf{s})$ divided by the execution time of $V(\mathbf{s})$. The left panel fixes the number of Brownian shocks at 1 and varies the number of state variables from 1 to 100, while the panel on the right fixes the number of state variables at 100 and varies the number of shocks from 1 to 100. In this example, V is represented by a two-layer neural network, and the executing times are computed 10,000 times on a mini-batch of 512 samples of the state space.

computational cost as measured by the wall-clock time scales less than one-for-one, as we compute the summation terms in Equation (8) in parallel.

2.2 The deep policy iteration algorithm

In this subsection, we show the update rules for the neural network parameters based on a generalized policy iteration. For ease of exposition, we make a few simplifying assumptions that can be easily relaxed. First, the update rules are based on the simplest version of the SGD, shown in Equation (2). Second, we alternate between exactly one step of policy evaluation and one step of policy improvement. Third, we use a quadratic loss function for the policy evaluation step.

Consider the class of standard optimal control problems in continuous time where infinitely lived agents face a Markovian decision process, with the vector of state variables $\mathbf{s} \in \mathcal{S} \subset \mathbf{R}^n$ subsuming all relevant information for decision-making. An agent chooses the policy $\mathbf{c}: \mathcal{S} \to \Gamma$ to maximize her lifetime expected utility:

$$V(\mathbf{s}_t) = \max_{\{\mathbf{c}_v\}} \mathbb{E}_t \left[\int_t^\infty e^{-\rho(v-t)} u(\mathbf{c}_v) dv \right]$$

$$\text{s.t.} \quad d\mathbf{s}_t = \mathbf{f}(\mathbf{s}_t, \mathbf{c}_t) dt + \mathbf{g}(\mathbf{s}_t, \mathbf{c}_t) d\mathbf{B}_t,$$

$$\mathbf{c}_t \in \Gamma(\mathbf{s}_t), \forall t \in [0, \infty),$$

where, at every point in the state space $\mathbf{s}_t$, the agent chooses controls $\mathbf{c}_t$ to maximize $V(\mathbf{s}_t)$ subject to the evolution of the state variables and a set of constraints on the controls $\Gamma(\mathbf{s}_t)$.

Under technical conditions, an intermediate step in the heuristic derivation of the associated HJB equation is

$$V(\mathbf{s}) = V(\mathbf{s}) + \max_{\mathbf{c} \in \Gamma(\mathbf{s})} \{\text{HJB}(\mathbf{s}, \mathbf{c}, V(\mathbf{s}))\} dt,$$

where

$$\text{HJB}(\mathbf{s}, \mathbf{c}, V) = u(\mathbf{c}) - \rho V + (\nabla_{\mathbf{s}} V)^\top \mathbf{f}(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{Tr}\left[\mathbf{g}(\mathbf{s}, \mathbf{c})^\top \mathbf{H}_{\mathbf{s}} V \mathbf{g}(\mathbf{s}, \mathbf{c})\right]$$

$$= u(\mathbf{c}) - \rho V + F''(0),$$

where F is the auxiliary function defined in Proposition 1. The solution to this problem is a pair of functions $V(\mathbf{s})$ and $\mathbf{c}(\mathbf{s})$ that satisfy at every point $\mathbf{s}$ in the state space, the following system of equations

$$\begin{aligned} 0 &= \text{HJB}(\mathbf{s}, \mathbf{c}(\mathbf{s}), V(\mathbf{s})), \\ \mathbf{c}(\mathbf{s}) &= \arg\max_{\mathbf{c} \in \Gamma(\mathbf{s})} \text{HJB}(\mathbf{s}, \mathbf{c}, V(\mathbf{s})). \end{aligned} \tag{10}$$

Representing the infinite-dimensional objects V and $\mathbf{c}$ on a computer requires an approximation using a finite set of parameters that we represent as

the vectors $\boldsymbol{\theta}_V$ and $\boldsymbol{\theta}_C$, respectively. A standard way of solving the problem in Equation (10) is to choose a finite subset of the state space $\{\mathbf{s}_i\}_{i=1}^I$ and parameterize the value and policy functions using as many parameters as there are states: $\mathbf{c}(\mathbf{s}_i;\boldsymbol{\theta}_C)=\boldsymbol{\theta}_{C,i}$ and $V_\pi(\mathbf{s}_i;\boldsymbol{\theta}_V)=\boldsymbol{\theta}_{V,i}$, where $\boldsymbol{\theta}_{V,i}$ is the ith entry of the vector $\boldsymbol{\theta}_V$ and $\boldsymbol{\theta}_{C,i}$ is the ith entry of the vector $\boldsymbol{\theta}_C$. With a slight abuse of notation, we can write the HJB error for state $\mathbf{s}_i$ as $\text{HJB}(\mathbf{s}_i;\boldsymbol{\theta}_C,\boldsymbol{\theta}_V)$. Under this approximation, functional equations become vector equations, and the problem can be exactly solved with policy iteration, as described in Section 1.3. In this case, the method consists of guessing initial $\boldsymbol{\theta}_V^0$ and $\boldsymbol{\theta}_C^0$ and constructing a sequence $\{\boldsymbol{\theta}_C^j,\boldsymbol{\theta}_V^j\}_{j\in\mathbb{N}}$ as follows:

$$\begin{aligned}\boldsymbol{\theta}_{C,i}^j &= \underset{\mathbf{c}\in\Gamma(\mathbf{s}_i)}{\arg\max}\,\text{HJB}(\mathbf{s}_i,\mathbf{c};\boldsymbol{\theta}_V^{j-1})\\ \boldsymbol{\theta}_{V,i}^j &= \boldsymbol{\theta}_{V,i}^{j-1}+\text{HJB}(\mathbf{s}_i;\boldsymbol{\theta}_C^j,\boldsymbol{\theta}_V^{j-1})\Delta t,\end{aligned} \tag{11}$$

until some stopping criterion is met.

Different from existing numerical methods that rely on a discretization of the state space and the iteration of Equation (11), we propose approximating the value function V and the policy $\mathbf{c}$ with a deep neural network and alternating between the following three steps, until a prespecified stopping criteria is met.

Step 1. Sampling Consider a random sample of points $\{\mathbf{s}_i\}_{i=1}^I$ in the state space. This mini-batch of size I can be sampled either from a uniform distribution between hypothesized bounds of the state space or from a guess (perhaps informed by previous iterations) about what the ergodic distribution looks like.

Step 2. Policy improvement The policy improvement step, illustrated in the second row of Equation (10), involves an optimization step for every state. This step of optimizing for every single state can be computationally very costly and is the driver of the third curse of dimensionality. Moreover, in general, this step cannot be solved exactly.

Consider then the following alternative approximate policy improvement strategy. For each state $\mathbf{s}_i$ in the mini-batch and starting from the initial guess $\mathbf{c}_{0,i}\equiv\mathbf{c}(\mathbf{s}_i;\boldsymbol{\theta}_C^{j-1})$, do one step of gradient descent on $-\text{HJB}(\mathbf{s}_i,\mathbf{c},\boldsymbol{\theta}_V^{j-1})$ using a learning rate of 1. The new control for each point in the mini-batch is

$$\mathbf{c}_{1,i}=\mathbf{c}_{0,i}+\nabla_{\mathbf{c}}\text{HJB}(\mathbf{s}_i;\mathbf{c}_{0,i},\boldsymbol{\theta}_V^{j-1}). \tag{12}$$

We can use these new values in Equation (12) as *targets* to train the policy network. The objective is to find $\boldsymbol{\theta}_C^j$ to minimize the quadratic loss function

$$\boldsymbol{\theta}_C^j=\underset{\boldsymbol{\theta}}{\arg\min}\,\mathcal{L}(\boldsymbol{\theta}),\ \text{where}$$

$$\mathcal{L}(\boldsymbol{\theta})=\frac{1}{2I}\sum_{i=1}^{I}||\mathbf{c}(\mathbf{s}_i;\boldsymbol{\theta})-\mathbf{c}_{1,i}||^2.$$

Since the gradient of the loss function $\mathcal{L}(\boldsymbol{\theta})$ is given by

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{I}\sum_{i=1}^{I}(\mathbf{c}(\mathbf{s}_i;\boldsymbol{\theta}) - \mathbf{c}_{1,i})^{\top}\boldsymbol{J}_{\boldsymbol{\theta}}\mathbf{c}(\mathbf{s}_i;\boldsymbol{\theta}),$$

where $\boldsymbol{J}_{\boldsymbol{\theta}}\mathbf{c}(\mathbf{s}_i;\boldsymbol{\theta})$ denotes the Jacobian of $\mathbf{c}(\mathbf{s}_i;\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, we can update $\boldsymbol{\theta}_C$ by taking one step along this gradient. Thus, an application of the one-step SGD evaluated at the starting point $\boldsymbol{\theta} = \boldsymbol{\theta}_C^{j-1}$ gives

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}^{j-1}) &= \frac{1}{I}\sum_{i=1}^{I}(\mathbf{c}_{0,i} - \mathbf{c}_{1,i})^{\top}\boldsymbol{J}_{\boldsymbol{\theta}}\mathbf{c}(\mathbf{s}_i;\boldsymbol{\theta}_C^{j-1}) \\ &= -\frac{1}{I}\sum_{i=1}^{I}\nabla_{\mathbf{c}}\mathrm{HJB}(\mathbf{s}_i, \mathbf{c}_{0,i}, \boldsymbol{\theta}_V^{j-1})^{\top}\boldsymbol{J}_{\boldsymbol{\theta}}\mathbf{c}(\mathbf{s}_i;\boldsymbol{\theta}_C^{j-1}) \qquad (13) \\ &= -\frac{1}{I}\sum_{i=1}^{I}\nabla_{\boldsymbol{\theta}_C}\mathrm{HJB}(\mathbf{s}_i, \boldsymbol{\theta}_C^{j-1}, \boldsymbol{\theta}_V^{j-1}),\end{aligned}$$

where the second row follows from Equation (12) and the last row from an application of the chain rule. Plugging Equation (13) into the update rule for gradient descent with learning rate η_C yields:

Policy Improvement

$$\boldsymbol{\theta}_C^{j} = \boldsymbol{\theta}_C^{j-1} + \eta_C\frac{1}{I}\sum_{i=1}^{I}\nabla_{\boldsymbol{\theta}_C}\mathrm{HJB}(\mathbf{s}_i, \boldsymbol{\theta}_C^{j-1}, \boldsymbol{\theta}_V^{j-1}). \qquad (14)$$

Step 3. Policy evaluation For the policy evaluation step, we present two alternative update rules. Each has advantages and disadvantages that are discussed below.

The first update rule is the analog of iterative policy evaluation in Equation (4). The *continuous-time Bellman target* is

$$V(\mathbf{s};\boldsymbol{\theta}^{j}) = V(\mathbf{s};\boldsymbol{\theta}_V^{j-1}) + \mathrm{HJB}(\mathbf{s}, \boldsymbol{\theta}_C^{j}, \boldsymbol{\theta}_V^{j-1})\Delta t. \qquad (15)$$

Given the sample $\{\mathbf{s}_i\}$, $\boldsymbol{\theta}_V^{j}$ minimizes the quadratic loss function

$$\boldsymbol{\theta}_V^{j} = \arg\min_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}), \text{ where}$$

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2I}\sum_{i=1}^{I}(V(\mathbf{s}_i;\boldsymbol{\theta}) - V(\mathbf{s}_i;\boldsymbol{\theta}_V^{j-1}) - \mathrm{HJB}(\mathbf{s}_i, \boldsymbol{\theta}_C^{j}, \boldsymbol{\theta}_V^{j-1})\Delta t)^2.$$

Since the gradient of the loss function $\mathcal{L}(\boldsymbol{\theta})$ is given by

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{I}\sum_{i=1}^{I}(V(\mathbf{s}_i;\boldsymbol{\theta}) - V(\mathbf{s}_i;\boldsymbol{\theta}_V^{j-1}) - \mathrm{HJB}(\mathbf{s}_i, \boldsymbol{\theta}_C^{j}, \boldsymbol{\theta}_V^{j-1})\Delta t)\nabla_{\boldsymbol{\theta}}V(\mathbf{s}_i;\boldsymbol{\theta}),$$

we can update $\boldsymbol{\theta}_V$ by taking one step along this gradient. Thus, an application of the one-step SGD evaluated at the starting point $\boldsymbol{\theta} = \boldsymbol{\theta}_V^{j-1}$ gives

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_V^{j-1}) = -\frac{\Delta t}{I} \sum_{i=1}^{I} \text{HJB}(\mathbf{s}_i, \boldsymbol{\theta}_C^j, \boldsymbol{\theta}_V^{j-1}) \nabla_{\boldsymbol{\theta}} V(\mathbf{s}_i; \boldsymbol{\theta}_V^{j-1}). \tag{16}$$

Plugging Equation (16) into the update rule for gradient descent with learning rate η_V yields

Policy Evaluation 1

$$\boldsymbol{\theta}_V^j = \boldsymbol{\theta}_V^{j-1} + \eta_V \frac{\Delta t}{I} \sum_{i=1}^{I} \text{HJB}(\mathbf{s}_i, \boldsymbol{\theta}_C^j, \boldsymbol{\theta}_V^{j-1}) \nabla_{\boldsymbol{\theta}_V} V(\mathbf{s}_i; \boldsymbol{\theta}_V^{j-1}). \tag{17}$$

An alternative to the policy evaluation step is the analog of direct policy evaluation in Equation (1.3.1). Directly minimizing the MSE of the Bellman residuals using SGD gives

Policy Evaluation 2

$$\boldsymbol{\theta}_V^j = \boldsymbol{\theta}_V^{j-1} - \eta_V \frac{1}{I} \sum_{i=1}^{I} \text{HJB}(\mathbf{s}_i, \boldsymbol{\theta}_C^j, \boldsymbol{\theta}_V^{j-1}) \nabla_{\boldsymbol{\theta}_V} \text{HJB}(\mathbf{s}_i, \boldsymbol{\theta}_C^j, \boldsymbol{\theta}_V^{j-1}). \tag{18}$$

In the machine-learning literature, methods that directly minimize the Bellman residuals are known to be slower than methods based on iterative policy evaluation. Furthermore, notice that the update rule in Equation (18) involves relatively costly third-order derivatives since it requires the gradient of the HJB residual. Nevertheless, residual methods are typically more stable than iterative policy evaluation when using nonlinear function approximation (Baird 1995). Therefore, as a rule of thumb, we recommend starting with the update rule in Equation (17), and switching to Equation (18) if the value function starts to diverge.

2.3 Hyperparameters

Note that a researcher has flexibility in how to implement such an algorithm. Design choices include the architecture of the networks (number of hidden layers and units), the optimization algorithm, the activation function, the learning rate, the number of steps for policy evaluation and policy improvement, and the sampling strategy. These are called *hyperparameters*. As with any numerical solution method, there are two ways of choosing the hyperparameters. The first one is to use a hyperparameter tuner software that searches for optimal values based on a given performance criterion.[18] The second way is to use previous work as a baseline and experiment with variations of that baseline. Since one of

[18] See, for instance, Liaw et al. (2018), Song et al. (2023), and Rapin and Teytaud (2018).

the contributions of this study is to establish such baselines for future work, we deliberately avoid automatic hyperparameter tuning because it is not necessary for our applications.

We use the same neural network architecture and hyperparameters for all applications in this paper. In particular, we use a three-layer neural network with 256 hidden units and layer normalization in the first layer, 128 hidden units in the second layer, and 64 units in the third layer. In our experience, such large networks have enough expressive power to accurately represent highly nonlinear functions with dozens of dimensions. For the policy functions, we use the ReLu activation function, which is one of the most commonly used activation functions in deep learning. For the value functions, we use a sigmoid linear unit (SiLu) activation function, which is similar to ReLu, but has the property of being twice continuously differentiable, as required by Ito's Lemma. For the SGD optimization of the policy evaluation step, we use the Adam optimizer with default hyperparameter values: learning rate = 10^{-3}, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. We find that using a smaller learning rate for the policy improvement step helps to prevent divergence, and initialize it at 10^{-4}. Both learning rates decrease by 1% every 15,000 iterations. We choose a batch size of 2,048, which is large enough to keep the GPU at 100 % utilization.

3. Applications

To showcase the broad applicability of our method, we solve three problems with a high degree of complexity in three core areas of finance, namely asset pricing, corporate finance, and portfolio choice. We start with the many-tree extension of the classical asset pricing model of Lucas (1978). We then consider a structural corporate finance model in the spirit of Hennessy and Whited (2007). Finally, we study a high-dimensional version of the portfolio choice problem of Campbell and Viceira (1999). While we focus on models with CRRA preferences and Brownian shocks to keep the exposition of the different applications as simple as possible, our method also works in more complex economies where investors have Epstein-Zin preferences and state variables are driven by jump-diffusion processes (see Appendix B).

3.1 Asset pricing

Consider first the two-tree economy of Cochrane, Longstaff, and Santa-Clara (2008), who extend the Lucas economy by adding another exogenously specified tree producing the same consumption good. Under restrictive assumptions, the authors derive closed-form expressions for the equilibrium objects, which we use to check the accuracy of the numerical solutions produced by our method. Later, we consider a richer version of the model with a large number of trees for which closed-form solutions are not available.

3.1.1 Two trees We keep the exposition of the benchmark model to its minimum and refer readers to Cochrane, Longstaff, and Santa-Clara (2008) for a detailed description of the model. In short, a representative consumer chooses a consumption stream to maximize the lifetime expected utility

$$V = \mathbb{E}\left[\int_0^\infty e^{-\rho t}\log(C_t)dt\right].$$

The aggregate consumption process $C=(C_t)_{t\geq 0}$ is the sum of the dividend streams D_{1t} and D_{2t} produced by the two trees. The exogenous dividend process $D_i=(D_{it})_{t\geq 0}$, with $i\in\{1,2\}$, follows a standard geometric Brownian motion:

$$\frac{dD_{it}}{D_{it}} = \mu dt + \sigma dZ_{it}.$$

The Brownian shocks Z_1 and Z_2 have instantaneous correlation equal to ϱ.

In this two-tree economy, the equilibrium quantities such as the short-term interest rate, dividend yield, expected return, and asset volatility are determined by a single state variable, namely, the dividend share $s_t = D_{1t}/(D_{1t}+D_{2t})$. Figure 4 compares the numerical solution produced by the DPI method and

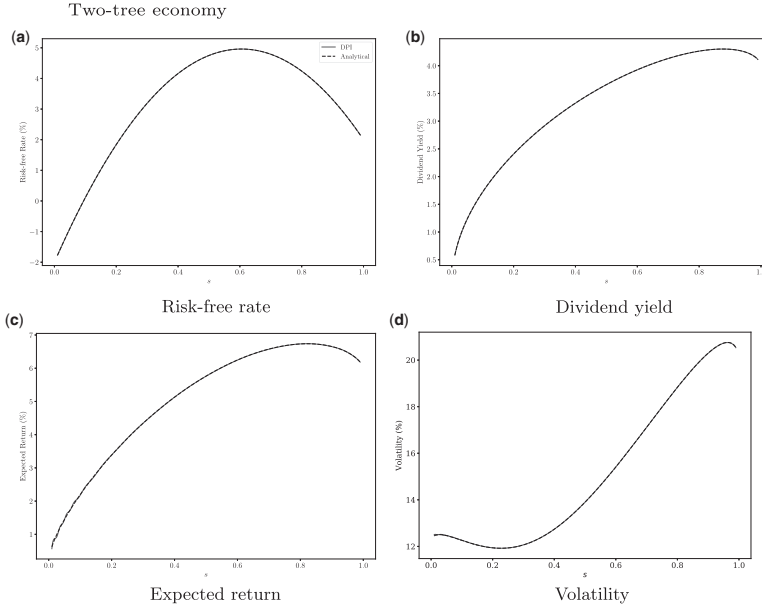

Figure 4
Two-tree economy
The figure shows the plots of the risk-free rate, dividend yield, expected return, and instantaneous volatility as a function of the first tree dividend share. The solid lines represent the numerical solutions, and the dashed lines represent the analytical solutions evaluated on the random test set. The values of the parameters are as follows: $\rho=0.04$, $\gamma=1$, $\varrho=-0.5$, $\mu_1=0.02$, $\mu_2=0.03$, $\sigma_1=0.2$, and $\sigma_2=0.3$. We use a neural network to approximate normalized asset prices $V=P\cdot(D_1+D_2)^{-\gamma}$. The iteration stops when the average error of the dividend yield is less than 10^{-5}.

the analytical solution of Cochrane, Longstaff, and Santa-Clara (2008) for the four equilibrium quantities mentioned above as a function of the state variable s_t.[19] As indicated, the high nonlinearities exhibited by these functions suggest that methods based on log linearization may fail to accurately capture these curvatures, resulting in inaccurate numerical solutions. The DPI method, in contrast, has no difficulty in capturing nonlinear dynamics due to its global nature and the flexibility of neural networks.

We use two measures to assess the accuracy of the numerical solution: (1) the absolute deviation of the numerical solution from the exact one and (2) the HJB residuals. Figure 5 shows the distribution of our two accuracy measures. To obtain these distributions, we randomly draw 10,000 values for s uniformly from $[0,1]$ and compute the value of the two measures for each draw. The panel on the left of Figure 5 shows the distribution of the absolute difference between the numerical and analytical solution for the dividend-yield, $\varepsilon_d(\mathbf{s}) = \log_{10}|d^{\text{numerical}}(\mathbf{s}) - d^{\text{analytical}}(\mathbf{s})|$. For conciseness, we only report the accuracy for the dividend yield as the results for the other variables are similar. We find that the average deviation is -5.04, with a standard deviation of 0.34, showing that the solution is accurate approximately up to the fifth decimal place.

Since $\varepsilon_d(\mathbf{s})$ requires the knowledge of the exact solution, this measure is restricted to economies for which closed-form solutions are available. For more complex economies where analytical solutions are not available, we consider a second measure of accuracy, namely, the HJB residuals. The HJB residuals correspond to the normalized deviations from the HJB equation, defined as $\varepsilon_{HJB}(\mathbf{s}) = \log_{10}\frac{|\text{HJB}(\mathbf{s},\mathbf{c}(\mathbf{s}))|}{V(\mathbf{s})}$, where $\text{HJB}(\mathbf{s},\mathbf{c}(\mathbf{s}))$ is given in Equation (10).[20] The panel on the right of Figure 5 shows the distribution of the HJB residuals. The distribution has a mean of -4.56 and a standard deviation of 0.56, once again showing that the solution has high accuracy. Combined, these results show that the distribution of HJB residuals is similar to the distribution of the absolute deviation errors, indicating that the two measures of accuracy are quantitatively similar.

3.1.2 Lucas Orchard The curse of dimensionality becomes apparent when we move from the two-tree Lucas economy of Cochrane, Longstaff, and Santa-Clara (2008) to the Lucas orchard economy of Martin (2013). The author generalizes the two-tree economy of Cochrane, Longstaff, and Santa-Clara (2008) by assuming the existence of N trees and by relaxing the log utility assumption on the representative agent's utility function. Martin (2013) provides semianalytical expressions for the equilibrium quantities as functions

[19] All numerical computations in the paper are done using a NVIDIA A100 GPU.

[20] HJB residuals can be interpreted as the continuous-time analog of the Euler equation errors commonly used in discrete-time models. For a discussion of the use of this metric in continuous-time settings, see Parra-Alvarez (2018).

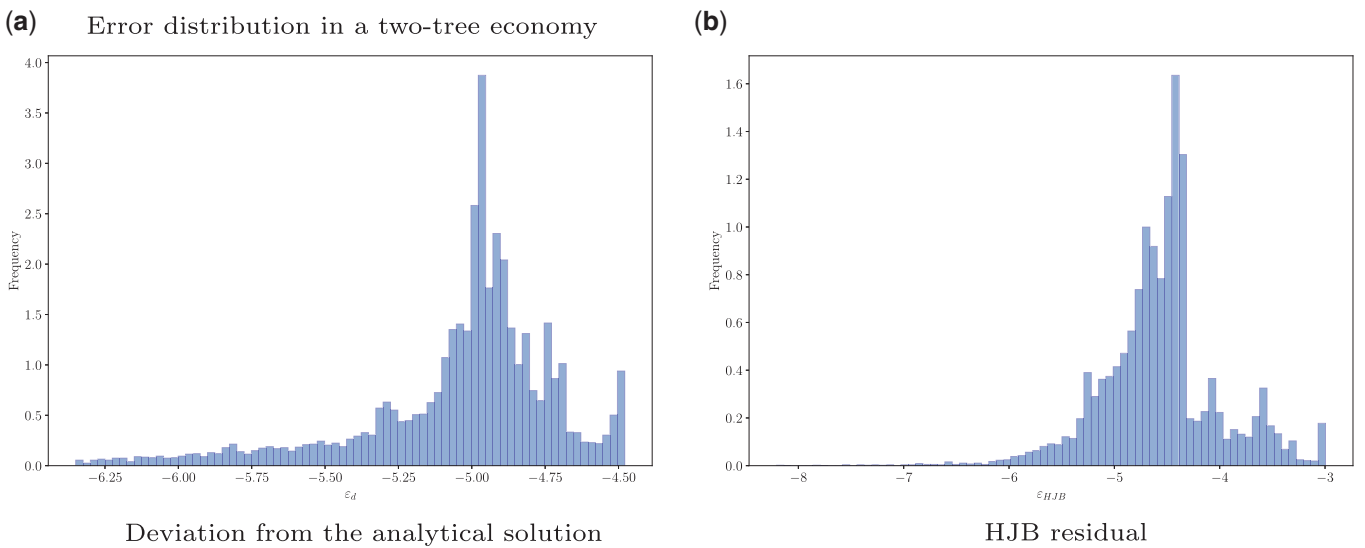

Figure 5
Error distribution in a two-tree economy
The left panel shows the distribution of the ($\log_{10}$) absolute difference between the numerical and the analytical solution of the dividend yield, while the right panel shows the ($\log_{10}$) HJB residuals for a test set of 10,000 randomly drawn points with $s \in [0, 1]$. The iteration stops when the average error is less than 10^{-5}.

of the $N-1$ dividend shares in the economy. However, the integral formulas are subject to a severe curse of dimensionality, which limits the applicability of the analytical results to setups with at most three or four trees.

To illustrate how the DPI method can alleviate the curse of dimensionality, we conduct the following experiment. Starting from an economy with two trees, we gradually increase the number of identical trees in the economy and solve for the equilibrium using the DPI algorithm. We consider two stopping criteria. First, we stop the iteration when an MSE lower than 10^{-8} is achieved. Second, we adopt a more stringent accuracy metric and stop the iteration when the 90th percentile of the squared errors is lower than 10^{-8}. Panel A of Figure 6 shows the time in minutes to compute the solution using the two criteria. The figure shows that the DPI method produces accurate solutions for problems with a high-dimensional state space in a timely manner. Moreover, raising the dimensionality of the problem or considering a more stringent accuracy measure does not substantially increase the time-to-solution. For instance, even in an economy with 100 trees, it takes less than a minute for the DPI algorithm to reach an MSE of 10^{-8}.

Panel B of Figure 6 shows the time-to-solution of the DPI method and the Smolyak method. The Smolyak method is arguably among the most widely used techniques in financial economics to tackle high-dimensional stochastic dynamic models.[21] Hence, it is important to compare how our method performs relative to it. We consider the Smolyak method of orders 2, 3, and 4, and we solve for the coefficients using the conjugate gradient method. We set the tolerance for the MSE to 10^{-3}, the highest threshold reached by all versions of the Smolyak method. The time-to-solution of the different versions of the Smolyak method increases rapidly with the number of trees in the economy, and the computer runs out of memory for orchards with 8, 12, and 26 trees for the methods of order 2, 3, and 4, respectively. In contrast, the DPI method is able to maintain high accuracy with a relatively low time-to-solution for economies with a much larger number of trees. This illustrates the ability of the DPI method to alleviate the curse of dimensionality relative to previously known methods.

3.1.3 Economic consequences of large N Next, we consider how the equilibrium objects vary the number of trees. We show that when the analysis is restricted to a small number of trees, because of either numerical limitations or for the sake of analytical tractability, important economic channels are overlooked. This leads to significantly different equilibrium outcomes, both quantitatively and qualitatively.

We illustrate this point by examining the equilibrium objects of a Lucas orchard with N trees for $N \in \{2,3,5,10,50\}$. The dividend process is the same

[21] In recent years, some notable contributions have increased the efficiency and accuracy of the Smolyak methods. See, for example, Judd et al. (2014), Brumm and Scheidegger (2017), and Brumm et al. (2022).

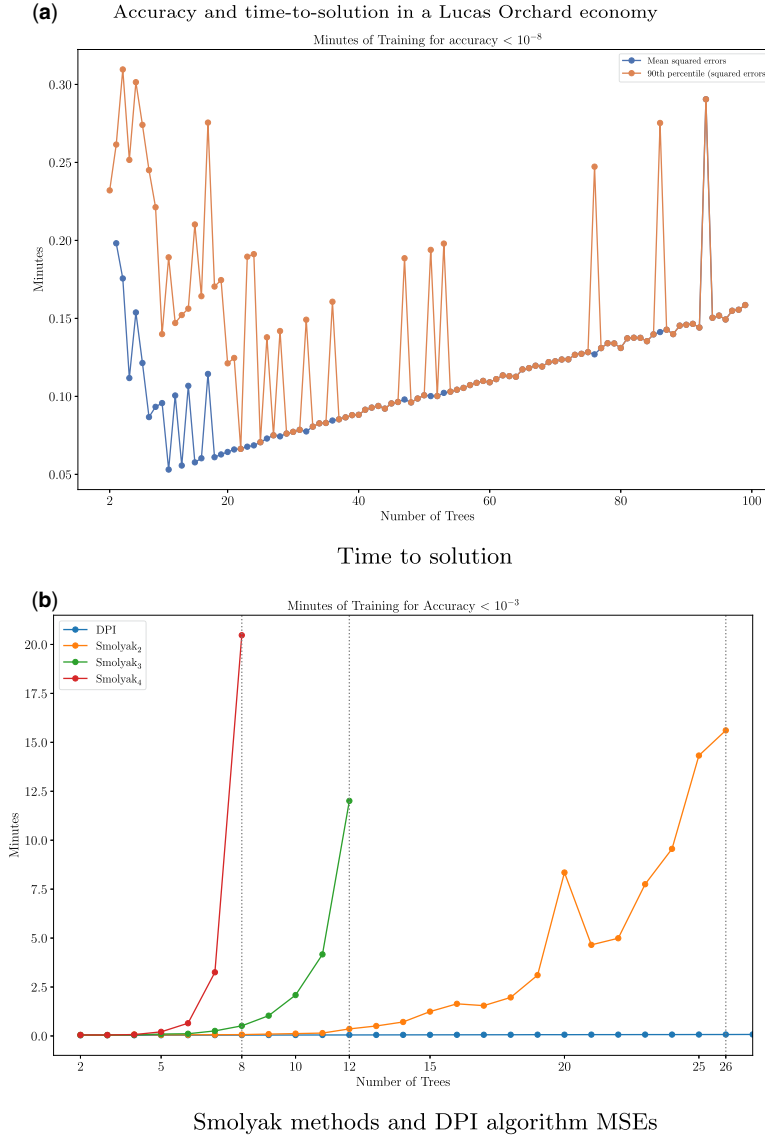

Figure 6
Accuracy and time-to-solution in a Lucas Orchard economy
Panel A shows the time-to-solution of the DPI algorithm, measured by the number of minutes required for a given metric to be less than 10^{-8}. The blue line represents the mean squared errors, and the orange line represents the 90th percentile of the squared errors. Panel B shows the time-to-solution of the DPI method and the Smolyak methods of orders 2, 3, and 4. The tolerance is set to 10^{-3}, which is the highest threshold reached by all the Smolyak methods. The parameter values are as follows: $\rho = 0.04$, $\gamma = 1$, $\varrho = 0.0$, $\mu = 0.015$, and $\sigma = 0.1$. The HJB errors are computed on a random sample of 2^{13} points from the state space.

for all trees, with volatility σ and pairwise correlation ϱ. When $N > 2$, we need to specify not only the share of the first tree s^1 but also the dividend share distribution of the remaining trees $(s^2, s^3, \ldots, s^N)$. To represent this high-dimensional object in a two-dimensional graph, we draw 10,000 values of $(s^2, s^3, \ldots, s^N)$, after being normalized to add up to one, from a symmetric Dirichlet distribution with concentration parameter α and report equilibrium quantities averaged over these draws in Figure 7.[22]

When $N = 2$, we recover the results of Cochrane, Longstaff, and Santa-Clara (2008). In this economy, the dividend yield and the interest rate respond strongly to changes in the share of the first tree s_t^1, as shown in panels A and B of Figure 7. The risk premium is positive, and the correlation between asset 1 returns and consumption is large, even as s^1 approaches zero, as shown in panels C and D. Similarly, consumption and return volatilities are also highly sensitive to changes in s^1, as shown in panels E and F.

Figure 7 shows that the results change substantially when N is relatively large *and* s^1 is small, an important case largely ignored by the literature. This case is particularly important because, when $N = 2$, either the economy is diversified and the trees are similar in size (i.e., there is no small asset as $s^1 \approx s^2 \approx 0.5$), or the economy has a small asset, but it is extremely underdiversified, with the larger tree being responsible for nearly all consumption. The graphs show that the three-tree economy ($N = 3$) analyzed by Martin (2013) experiences similar drawbacks, albeit to a lesser extent. In contrast, when N is large, we can analyze the behavior of a small asset ($s^1 \approx 0$) in an economy that is still reasonably diversified, where no single tree represents the entirety of consumption.

For example, consider the case in which $N = 50$, and the dividend share of the first risky asset is in the range $0 < s^1 < 20\%$. Note that the dividend yield and interest rate barely move as the dividend share of the first asset s^1 changes. The reason is that aggregate consumption volatility is roughly insensitive to this state variable in this region as the other 49 trees still provide enough diversification.[23] In the absence of the effects on aggregate volatility, movements in interest rates and in the dividend yield are muted. In stark contrast, when $N = 2$, a reduction of s^1 from 20% to almost 0% substantially increases consumption volatility, resulting in significantly lower dividend yield and risk-free rate.

An important economic insight derived from Figure 7 is that when the economy has many trees *and* the dividend share s^1 is relatively small, the behavior of the economy resembles a fully diversified economy (horizontal red

[22] When $\alpha \approx 1$, the sampled dividend shares $(s_t^2, s_t^3, \ldots, s_t^N)$ are relatively dispersed. For larger values of α, the sampled dividend shares become more concentrated around the center of the simplex, and the draws tend to be similar to each other, consistent with the economy being more diversified.

[23] Naturally, as s^1 approaches 1, all economies behave similarly as the economy becomes concentrated on the first risky asset, regardless of the value of N.

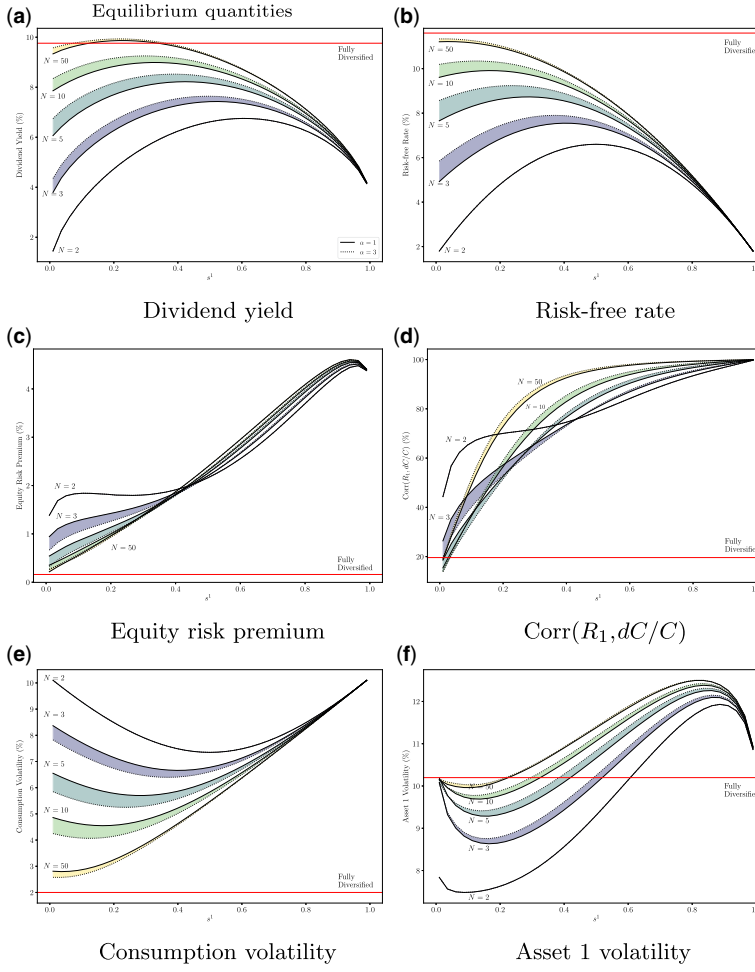

Figure 7
Equilibrium quantities
We discretize the interval $(0,1)$, representing the domain of the dividend share of the first risky asset s^1, into 100 equal parts. For each given point in the grid, we draw 10,000 samples of the remaining $N-1$ dividend shares $(s^2,s^3,\ldots,s^N)$ from a symmetric Dirichlet distribution with parameter α. With 10,000 samples for each point s^1 in the grid, we then compute the equilibrium quantities by evaluating the trained neural network model at each point in space $(s^1,s^2,\ldots,s^N)$ and averaging the result across the samples. We repeat this process for different levels of the concentration parameter α in the interval $[1,3]$. The remaining parameter values are as follows: $\rho=0.04$, $\gamma=4$, $\varrho=0.04$, $\mu=0.02$, $\sigma=0.1$, and $\sigma_{agg}=0.02$.

line) where consumption is exposed to only an aggregate shock.[24] Moreover, our results provide a quantitative assessment of how *fast* the equilibrium

[24] In the fully diversified economy the process for consumption is $dC_t/C_t=\mu dt+\sigma_{agg}dZ_t$, and dividend j follows the process $dD_{j,t}/D_{j,t}=\mu dt+\sigma dZ_{j,t}$, where $dZ_t dZ_{j,t}=\varrho$. We set $\sigma^2_{agg}=\varrho\sigma^2$ (i.e, the minimum consumption variance in the Lucas orchard as $N\to\infty$), corresponding to the nondiversifiable risk in this economy.

outcomes converge to this fully diversified benchmark. Even for moderately diversified economies, with $N=5$ or $N=10$, the market-clearing effects emphasized by Cochrane, Longstaff, and Santa-Clara (2008) get substantially attenuated, as seen, for instance, in panels C and F of Figure 7.

Martin (2013) argues that the positive risk premium of a small asset ($s^1 \approx 0$) is due to the high covariance of its valuation ratio with aggregate cash flows. As Figure 7 shows, these effects are greatly attenuated when there is sufficient diversification in the economy (e.g., $N=50$) and disappear when the economy is fully diversified. Thus, by lifting the numerical restrictions that had impeded the literature's ability to analyze the behavior of small firms in well-diversified economies, the DPI method reveals that the positive risk premium of a small asset is a byproduct of a severely underdiversified economy.

3.2 Corporate finance

As our next application, we consider a canonical corporate finance model. Even though the model can be solved using standard numerical techniques, we illustrate how the DPI algorithm's ability to handle large state spaces can be used by researchers to perform a *global sensitivity analysis*. This is an important step in showing how different moments in the data are informative about specific parameters in the model in a transparent way.[25] Moreover, by considering a nonconvex optimization problem, we illustrate how the DPI method seamlessly handles severe nonlinearities in the solution, such as multiple kinks.

3.2.1 Model environment We present a simplified version of the model by Hennessy and Whited (2007) that includes costly equity issuance and investment adjustment costs. To simplify the exposition, we abstract from taxes and a corporate debt decision. We assume that there is a firm with operating profits following a standard Cobb-Douglas production function $\pi(k_t, z_t) = z_t k_t^{\alpha}$, with elasticity $\alpha \in (0,1)$. Two state variables that drive operating profits: total factor productivity (TFP hereafter) z_t and the capital stock k_t. TFP follows an Ornstein-Uhlenbeck process in logs:

$$d\ln z_t = -\theta \ln z_t dt + \sigma dW_t, \text{ with } \theta, \sigma > 0.$$

Capital accumulates according to $\frac{dk_t}{k_t} = (i_t - \delta)dt$, where i_t represents the firm's investment rate and δ is the depreciation rate. Investment is subject to quadratic adjustment costs, $\Lambda(k_t, i_t) = 0.5\chi k_t i_t^2$, with $\chi > 0$.

[25] For a discussion of the role of transparency in structural research, including its connection with sensitivity analysis, see Andrews, Gentzkow, and Shapiro (2020).

As in Hennessy and Whited (2007), we assume that raising external equity is costly, and equity issuance is subject to the linear cost $\lambda > 0$. Since the firm's operating profits net of investment costs is $D_t^* = z_t k_t^\alpha - \left(i_t + 0.5\chi i_t^2\right)k_t$, the firm's dividend policy is given by

$$D_t = D_t^*(1 + \lambda \mathbb{1}_{D_t^* < 0}). \tag{19}$$

Equation (19) shows that the firm pays a unit cost λ if it decides to issue equity (i.e., if $D_t^* < 0$).

Given a discount rate $r > 0$, the firm's problem can be written as follows:

$$V(k, z; \Phi) = \max_{\{i_t\}_0^\infty} \mathbb{E}\left[\int_0^\infty e^{-rt} D_t dt\right], \tag{20}$$

subject to Equation (19), the law of motion of TFP and capital, the initial conditions $k_0 = k$ and $z_0 = z$, and a vector of model parameters Φ. Notice that, contrary to common practice in the literature, we explicitly state the dependence of the value function $V(k, z; \Phi)$ on the model parameters Φ to emphasize that the parameter choices alter the solution of the model.

This dynamic corporate finance model helps illustrate different aspects of the DPI algorithm. First, unlike the endowment economy considered in Section 3.1, the dynamics of one of the state variables is endogenous since the investment decision determines capital accumulation. In this case, we need to approximate both the value and policy functions with neural networks. Moreover, while only the policy evaluation step of the DPI method was needed in the Lucas orchard economy, a problem with an endogenous state variable requires both policy evaluation and policy improvement to compute the solution. Second, the cost of issuing equity creates kinks in the policy functions. Because kinks are a ubiquitous feature in a wide class of models with occasionally binding constraints or transaction costs, it is important to assess how the solution method performs in such a case.[26]

Figure 8 shows the policy functions $D(k, z)$ and $i(k, z)$ as a function of k for different values of z. The colored lines represent the solution obtained using the DPI method, while the black dashed lines represent the solution obtained using an implicit finite-differences scheme with a very fine grid, which serves as our benchmark. From the graphs, we observe that the solution using the DPI method closely tracks the solution obtained using finite differences. The accuracy of the solution is also demonstrated by the low $\log_{10}$ RMSE of the HJB residuals, which amounts to -5.

[26] The fact that the solution has kinks implies that a classical solution to the HJB equation for the firm's problem in Equation (20) does not exist and we instead look for a *viscosity solution* to the HJB. For a discussion of viscosity solutions, see Crandall (1995) and Achdou et al. (2022).

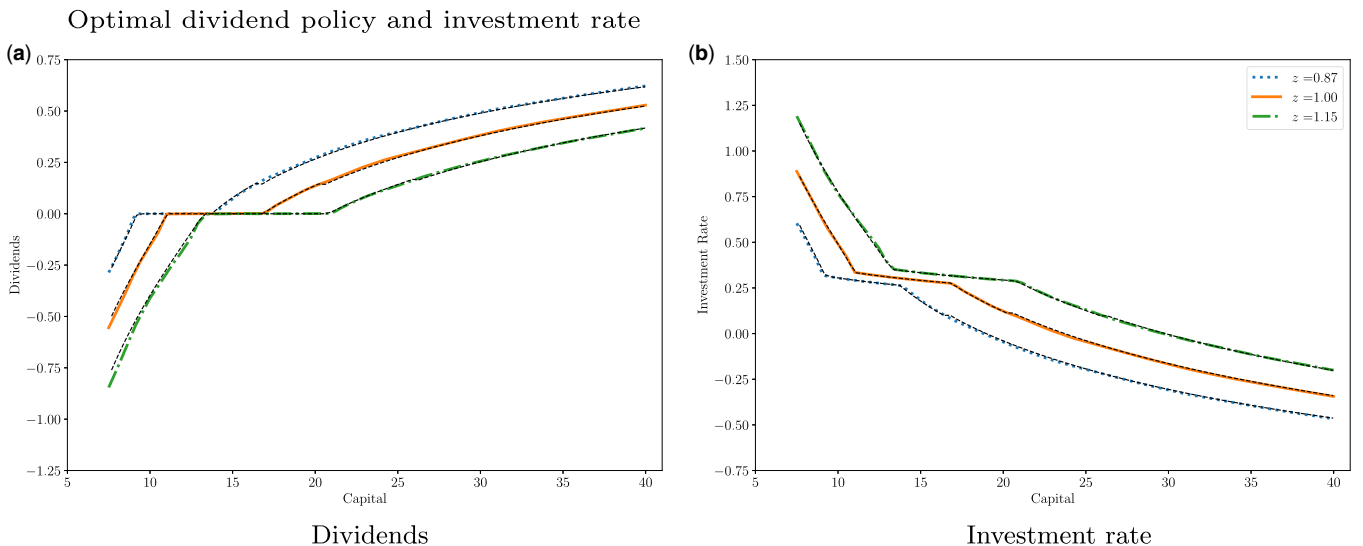

Figure 8
Optimal dividend policy and investment rate
The figure shows the plots of the optimal dividend policy and optimal investment rate as a function of the capital stock for different values of TFP. The colored lines (dotted, dash-dotted, and solid lines) represent the solution using the DPI method, and the black dashed lines represent the solution using an upwind finite differences method with 23,001 grid points (451 for capital, 51 for TFP). The RMSE of the HJB residuals for the DPI solution is 2.0×10^{-5}, computed on a random sample of $8,192,000$ observations (2^{14} parallel simulations of size $2,000$) sampled from the ergodic distribution. The value of the parameters are as follows: $\delta = 0.1$, $\alpha = 0.55$, $\lambda = 0.059$, $\theta = 0.26$, and $\sigma_z = 0.123$. The network takes as inputs the states (k, z) and the vector of model parameters Φ. The network was trained in approximately 1 hour.

The high accuracy level of the solution produced by the DPI algorithm is particularly noteworthy because of the presence of kinks in the firm's optimal dividend policy. As shown in Figure 8, the dividend policy can be divided into three regions. When the firm has a large initial capital stock k, it pays positive dividends. When the initial capital stock is small, the firm issues equity. However, for intermediate levels of capital, the firm neither pays dividends nor issues equity, creating *an inaction region*. The differences in the firm's payout policy in each region give rise to kinks in the optimal dividend policy function. Nonetheless, these nonlinearities do not pose a challenge for the DPI algorithm, which accurately captures both the region of inaction for dividends and the corresponding kinks.

3.2.2 Global sensitivity analysis and universal value functions For a given parametrization, the previous model can be easily solved using a standard numerical method, such as finite differences. However, since the model's results might be dependent on the chosen parametrization, we are often interested in the solution for different parameter values to check the robustness of our findings to changes in the parameter space in the context of structural work.[27] Thus, global sensitivity analysis is critical to identify which moments in the data are particularly informative about each parameter. However, because calibration or structural estimation may require solving the model for a large number of parameter values, sensitivity analysis can become computationally very costly and impractical in many cases.

To overcome the high computational cost of performing sensitivity analysis or structural estimation, we exploit the ability of the DPI algorithm to handle high-dimensional problems and include the model parameters as inputs to the neural network, as suggested by our formulation in Equation (20). In our experiment, this increases the computational cost only slightly, but once the network is trained, the solution is available for any point in the state and parameter spaces.[28]

In the literature on deep-reinforcement learning (Schaul et al. 2015), approximators similar to $V(k,z;\Phi)$ are known as *universal value functions* (UVFs hereafter).[29] In our experiment, the UVF depends on the state variables (k,z) and the vector of parameters $\Phi=(\lambda,\delta,\alpha,\theta,\sigma)$, for a total of seven variables.[30] The proposed approach allows us to obtain at once the solution for an entire *class* of models.

[27] A recent literature emphasizes the importance of sensitivity analysis in the context of structural analysis. See, for example, Andrews, Gentzkow, and Shapiro (2017), Armstrong and Kolesár (2021), and Catherine et al. (2022).

[28] We thank an anonymous referee for pointing out this fact to us.

[29] See Norets (2012) for an early application of this approach in a discrete-choice setting.

[30] To ease exposition and limit the number of results to report, we fix the values of r and χ. It is straightforward to include these parameters and perform sensitivity analysis with respect to them.

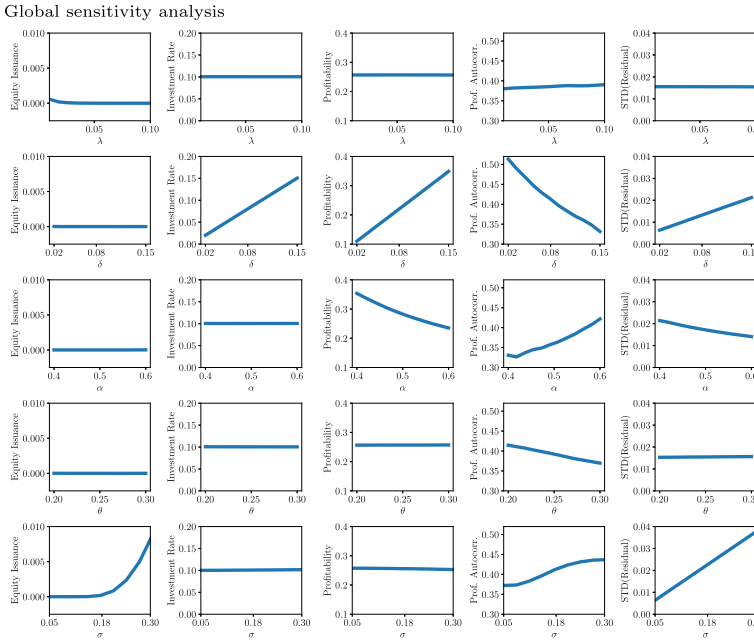

Figure 9
Global sensitivity analysis
The figure shows the plots of the following moments as a function of the parameters: (a) average equity issuance: $\mathbb{E}[\min\{D_t,0\}]$, (b) average investment rate: $\mathbb{E}[i_t]$, (c) average profitability: $\mathbb{E}[p_t]$, $p_t \equiv \pi(k_t,z_t)/k_t$, (d) annual autocorrelation of profitability: the slope coefficient of the regression $p_{t+1} = \alpha + \beta p_t + \sigma_\epsilon \epsilon_{t+1}$, and (e) the volatility of future profitability conditional on current profitability: σ_ϵ. The model solution is obtained by approximating value and policy functions by neural networks including the vector of parameters as inputs. For each column, we fix the parameters at the baseline values and then vary each parameter individually. The moments are computed by simulating 2^{12} economies in parallel for 2,000 periods, after dropping 1,000 burn-in periods. The SDE is simulated using the Euler method with a time step of 0.05.

Figure 9 shows the results of the global sensitivity analysis for our version of the Hennessy and Whited model. The figure shows selected moments of the equilibrium variables as a function of the parameters $\Phi = (\lambda, \delta, \alpha, \theta, \sigma)$. As illustrated, the average profitability is sensitive to α, while the average investment rate is particularly sensitive to δ. Note that the sensitivity of the moments to the parameters varies depending on the region of the parameter space. For example, for low-volatility firms, average equity issuance is relatively insensitive to σ, while for high-volatility firms, equity issuance is highly sensitive to σ. This particular feature of the solution could not be uncovered using a local sensitivity measure, such as the one proposed by Andrews, Gentzkow, and Shapiro (2017). The authors recommend that a local measure of the sensitivity of estimated parameter values to moments should be reported along with the results of a structural estimation.

In a sense, the sensitivity analysis we propose is a global version of their measure, as it allows one to assess how parameters affect moments not only in

the neighborhood of the estimated parameters but also for parameters far from their estimated values.[31]

To check the accuracy of the UVF approach, we compute the moments for 100 random draws from the parameter space and compare the solutions produced by a finite-difference method with a fine grid (our proxy for the exact solution) and our UVF approximator. Figure 10 shows the R^2 values for a regression comparing the moments computed with the DPI method with those computed with finite differences. The resultant R^2 is very close to one for all moments, and the moments computed with the DPI method and finite differences are very similar.

In summary, this exercise shows that the DPI method can be useful even when the model in question has a small number of state variables. In addition, while we have focused exclusively on the important topic of global sensitivity analysis, it is worth noting that similar methods can be used for structural estimation. For example, Duarte (2018) builds on the methods of this paper to show that UVFs can be used to efficiently estimate structural models.

3.3 Portfolio choice

In this section, we consider a high-dimensional version of the portfolio problem of Campbell and Viceira (1999) with time-varying expected returns. We first demonstrate our method's ability to provide accurate solutions. Since closed-form solutions are typically not available for these highly nonlinear problems, we propose a new method to test the accuracy of the DPI solution which consists of reverse engineering a portfolio problem with a known solution in a high-dimensional space. We then consider an empirically motivated portfolio problem with multiple risky assets and realistic return dynamics.

3.3.1 Reverse engineering a portfolio problem Model environment. Consider the problem of an investor with CRRA utility function who must choose the consumption policy C_t and the fraction of wealth invested in a (single) risky asset α_t, for given exogenous processes for the interest rate r_t and the risk premium ξ_t, in order to maximize her expected utility function:

$$V(W,\mathbf{x}) = \max_{\{C_t,\alpha_t\}_0^\infty} \mathbb{E}_0\left[\int_0^\infty e^{-\rho t}\frac{C_t^{1-\gamma}}{1-\gamma}dt\right], \tag{21}$$

subject to the wealth dynamics

$$dW_t = [(r_t+\alpha_t\xi_t)W_t - C_t]dt + \alpha_t W_t \sigma_{\mathbf{r}} d\mathbf{Z}_t.$$

[31] A similar approach to performing a global sensitivity analysis was recently proposed by Scheidegger and Bilionis (2019); Kase, Melosi, and Rottner (2022); Catherine et al. (2022). Similar to Duarte (2018), Catherine et al. (2022) construct moment networks that produce predicted moments as functions of the model parameters. The authors propose constructing a large data set of model parameters and their corresponding moments by solving a model tens of thousands of times. Alternatively, one can leverage the DPI method to construct the same data set by including the parameters as inputs to the network and solving the model only once.

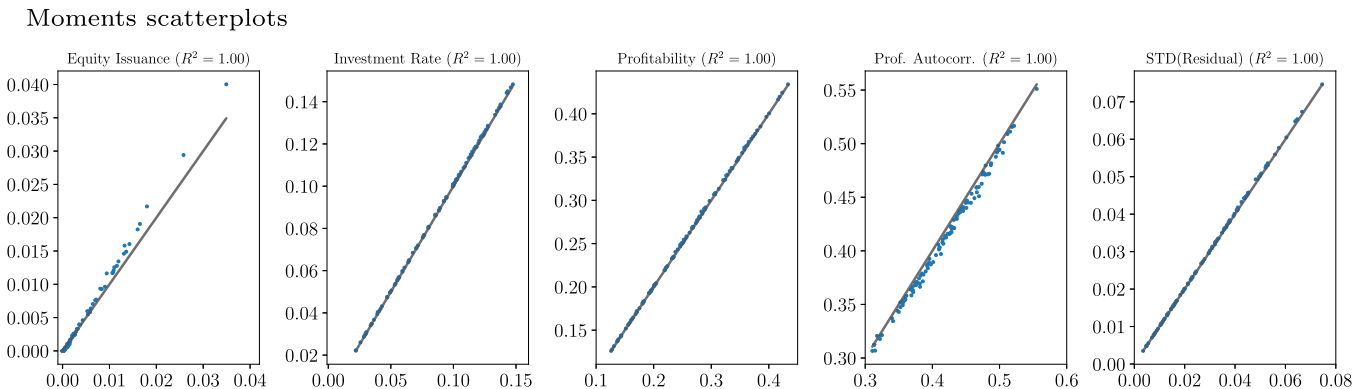

Figure 10
Moments scatterplots
The figure shows the scatterplots of moments computed using the DPI method, as described in Figure 9, against the solution using an upwind finite-differences method with 23,001 grid points (451 points for capital and 51 points for TFP). For the computation of the model-implied moments using the finite differences solution, we interpolate the points outside the grid with nearest-neighbor interpolation.

Here, $\mathbf{Z}_t$ is an $(N+1)$-dimensional Brownian motion, and $\sigma_{\mathbf{r}}$ is a constant $(N+1)$-dimensional (row) vector. The risk-free rate $r_t = r(\mathbf{x}_t)$ and the risk premium $\xi_t = \xi(\mathbf{x}_t)$ are assumed to be time-varying and driven by an N-dimensional state variable $\mathbf{x}_t$, with dynamics given by

$$d\mathbf{x}_t = \mu_{\mathbf{x}}(\mathbf{x}_t)dt + \sigma_{\mathbf{x}}(\mathbf{x}_t)d\mathbf{Z}_t, \tag{22}$$

where $\mu_{\mathbf{x}}(\mathbf{x}_t)$ is an N-dimensional vector and $\sigma_{\mathbf{x}}(\mathbf{x}_t)$ is an $N \times (N+1)$ matrix.

The vector $\mathbf{x}_t$ represents state variables that capture return predictability. It can include financial measures, such as the dividend-yield, the term spread, the investment-capital ratio of Cochrane (1991), the consumption-wealth ratio *cay* of Lettau and Ludvigson (2001), and the accounting growth measures of Daniel and Titman (2006).[32] We are interested in finding the optimal portfolio share $\alpha(\mathbf{x}_t)$ and the consumption policy $C(\mathbf{x}_t)$, given the dynamics of the predictors $\mathbf{x}_t$ in Equation (22), the risk-free rate $r(\mathbf{x}_t)$, and the risk premium $\xi(\mathbf{x}_t)$.

Reverse engineering. Since a closed-form solution to the previous problem is typically not available, we propose to reverse engineer the functions $r(\mathbf{x})$ and $\xi(\mathbf{x})$ to achieve any desired solution policies $\alpha(\mathbf{x})$ and $C(\mathbf{x})$. We can then use the DPI method with the reverse-engineered functions $r(\mathbf{x})$ and $\xi(\mathbf{x})$ to solve this high-dimensional portfolio problem and compare the solution of the algorithm with the known functions $\alpha(\mathbf{x})$ and $C(\mathbf{x})$ that were initially specified by the investigator.

To illustrate the proposed procedure, consider first a simple transformation of the consumption policy C_t that simplifies our exposition. By writing the value function in Equation (21) as $V(W,\mathbf{x}) = \phi(\mathbf{x})\dfrac{W^{1-\gamma}}{1-\gamma}$, where $\phi(\mathbf{x})$ is a value-function shifter to be determined, the first-order condition implies that $C_t = \phi(\mathbf{x}_t)^{-\frac{1}{\gamma}} W_t$. Since there is a one-to-one mapping between the consumption policy and the value-function shifter, giving a functional form to the value-function shifter $\phi(\mathbf{x}_t)$ is equivalent to modeling the consumption policy $C_t = C(\mathbf{x}_t)$ and vice versa. Moreover, the dynamics of the value-function shifter ϕ_t can be easily obtained by a simple application of Ito's lemma:

$$\frac{d\phi_t}{\phi_t} = \mu_\phi(\mathbf{x}_t)dt + \sigma_\phi(\mathbf{x}_t)d\mathbf{Z}_t.$$

Suppose now that the functional form of $\phi(\mathbf{x})$ and $\alpha(\mathbf{x})$ are known and set exogenously by the investigator. The functions $\xi(\mathbf{x})$ and $r(\mathbf{x})$ can be derived from the investor's optimality conditions and the investor's HJB equation, respectively, which yield the following expressions

[32] For analyses and reviews of the empirical performance of market return predictors, see Welch and Goyal (2008), Koijen and Van Nieuwerburgh (2011), and Lewellen (2015).

$$\xi(\mathbf{x}) = \gamma\,||\sigma_{\mathbf{r}}||^2\alpha(\mathbf{x}) - \sigma_\phi(\mathbf{x})\sigma_{\mathbf{r}}^\top, \tag{23}$$

$$r(\mathbf{x}) = \frac{\rho}{1-\gamma} + \frac{\gamma\,||\sigma_{\mathbf{r}}||^2}{2} - \left(\frac{\gamma\,\phi(\mathbf{x})^{-\frac{1}{\gamma}}}{1-\gamma} + \alpha(\mathbf{x})\xi(\mathbf{x}) + \sigma_\phi(\mathbf{x})\sigma_{\mathbf{r}}^\top + \frac{\mu_\phi(\mathbf{x}) + 0.5||\sigma_\phi(\mathbf{x})||^2}{1-\gamma}\right). \tag{24}$$

Thus, the expressions in Equations (23) and (24) allow us to obtain the values of $\xi(\mathbf{x})$ and $r(\mathbf{x})$ associated with any given value-function shifter $\phi(\mathbf{x})$ and portfolio share $\alpha(\mathbf{x})$.

We use this procedure to test the ability of the DPI algorithm to produce accurate solutions in high-dimensional portfolio-choice problems. Rather than choosing the functions ϕ and α based on economic considerations, we select functional forms that are known to be challenging for standard methods to approximate. We consider an empirically motivated case below. More specifically, for the value function shifter $\phi(\mathbf{x})$, we choose a multivariate version of the Runge function

$$\phi(\mathbf{x}) = \frac{1}{1 + \frac{25}{N}\sum_{j=1}^{N} x_i^2}, \tag{25}$$

which is typically used in numerical analysis to illustrate the difficulties of interpolation with polynomials.[33] For the portfolio share, we consider a highly nonlinear function capable of generating rich patterns for the relationship between the portfolio share and a given predictor. This is an important feature given the variety of predictors proposed in the literature. We model $\alpha(\mathbf{x})$ as

$$\alpha(\mathbf{x}) = \left|\sin\left(\sum_{j=1}^{N} x_j^2\right)\right|. \tag{26}$$

In our numerical exercise, we set the number of predictors to $N = 10$ so that it is in the ballpark of the number of predictors in the "kitchen sink" regression of Welch and Goyal (2008). We set $\mu_{\mathbf{x}}(\mathbf{x}) = -0.45\mathbf{x}$ and $\sigma_{\mathbf{x},i}(\mathbf{x}) = 0.1\mathbf{e}_i$, where $\sigma_{\mathbf{x},i}(\mathbf{x})$ denotes the ith row of $\sigma_{\mathbf{x}}(\mathbf{x})$ and $\mathbf{e}_i$ is the $(N+1)$th dimensional canonical basis vector in the ith direction. Therefore, the predictors follow uncorrelated Ornstein–Uhlenbeck processes with a volatility of 10% and a half-life of roughly 1.5 years, which is the average persistence of the different predictors reported in Gârleanu and Pedersen (2013).[34] We set $\sigma_{\mathbf{r}} = 0.2e_{N+1}$ so that return

[33] For a discussion of the Runge function and the corresponding challenges of approximating this function numerically, see, for example, Epperson (1987).

[34] The assumption that the predictors are uncorrelated can be interpreted as an extreme form of a shrinkage estimator applied to the variance-covariance matrix. For the importance of covariance shrinkage in portfolio optimization, see Ledoit and Wolf (2004) and Pedersen, Babu, and Levine (2021).

volatility is 20% and return innovations are uncorrelated with the predictors. We assume that the risk aversion coefficient is $\gamma = 2$ and the time-preference parameter is $\rho = 0.04$.

Numerical results. Figure 11 shows the value-function shifter and the portfolio share, respectively, as a function of the first predictor, x_1, for different values of the remaining predictors. The colored lines represent the solution obtained with the DPI method, and the black dashed lines correspond to the exact solution, as given by Equations (25) and (26). The approximate solution closely tracks the exact solution, and the $\log_{10}$ RMSE of the HJB residuals is -3, indicating that the solution is sufficiently accurate. Note that the solutions obtained with the DPI method for the value-function shifter $\phi(\mathbf{x})$ and the portfolio share $\alpha(\mathbf{x})$ do not exhibit the type of oscillations typically found in polynomial approximations of these functions. In addition, the portfolio share $\alpha(\mathbf{x})$ shows a rich pattern of behavior depending on the value of predictors 2 through 10. The functions can be V-shaped, increasing, or decreasing as a function of x_1 depending on the value of the remaining predictors $\mathbf{x}_{-1}$. Despite this wide range of behavior, the DPI method is able to accurately represent all these curves.

To further assess the accuracy of the solution, we consider a random sample of points drawn from the state space and compare the exact and approximate solutions. Figure 12 shows a scatter plot of the solution obtained using the DPI method against the exact solution. The points line up closely over the 45° degree line and the R^2 of the two regressions are essentially one.

3.3.2 Portfolio choice with realistic dynamics In this final application, we use the DPI algorithm to solve a high-dimensional portfolio choice problem calibrated with realistic asset pricing dynamics in which the expected returns on different asset classes are driven by several macro-finance variables.

Problem description. We depart from the portfolio problem discussed in Section 3.3.1 in three important dimensions. First, we build on the state-of-the-art affine model of Jiang et al. (2019) to discipline the evolution of expected returns. In particular, we consider a flexible model for the state-price density (SPD) that accurately prices stocks, nominal bonds, and inflation-protected bonds. Second, we assume that the investor has recursive preferences with a risk aversion coefficient γ and an elasticity of intertemporal substitution (EIS) ψ. Third, the investor has access to five risky assets in addition to a risk-free money market account. The vector of risky assets includes stocks, long- and medium-term nominal bonds, and long- and medium-term real bonds.

Expected returns are driven by a $N \times 1$ vector of state variables $\mathbf{x}_t \in \mathbb{R}^N$. The vector of state variables evolves according to a multivariate Ornstein-Uhlenbeck process:

$$d\mathbf{x}_t = -\Phi \mathbf{x}_t + \sigma_{\mathbf{x}} d\mathbf{Z}_t, \tag{27}$$

where $\Phi \in \mathbb{R}^{N \times N}$ is a matrix of coefficients and $\sigma_{\mathbf{x}} \in \mathbb{R}^{N \times N}$ is a matrix of loadings on the $N \times 1$ Brownian motion $\mathbf{Z}_t$. The real risk-free rate $r(\mathbf{x}_t)$ and the

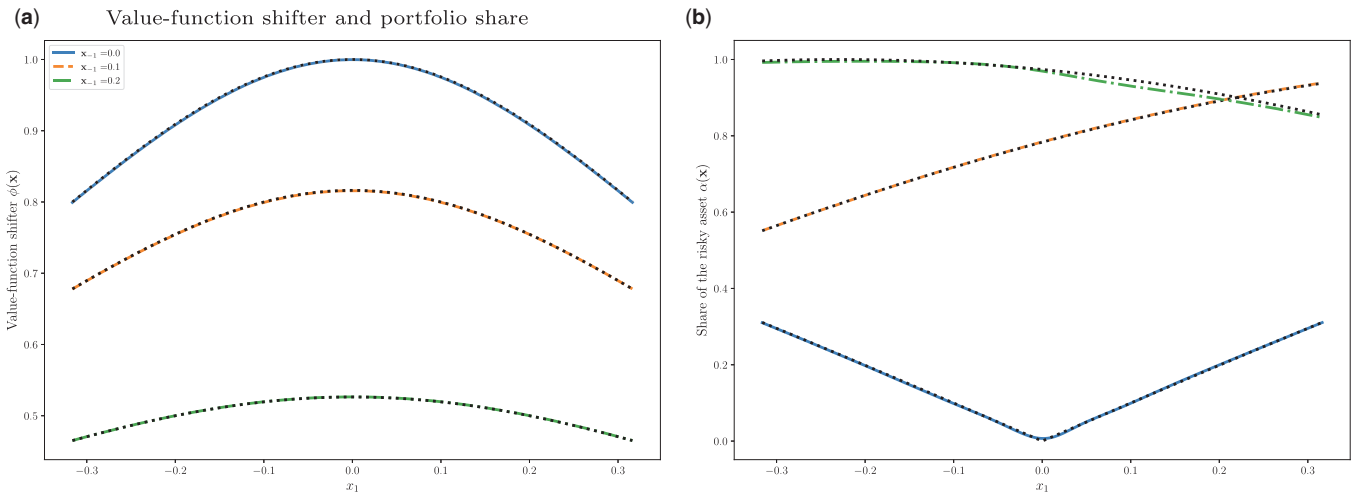

Figure 11
Value-function shifter and portfolio share
The figure shows the plots of the value-function shifter and the portfolio share as a function of the first predictor, x_1, given the value of the remaining predictors, $\mathbf{x}_{-1}$, and $W=1$. The colored lines (dotted, dash-dotted, and solid lines) represent the solution using the DPI method, and the black dashed lines correspond to the exact solution given by Equations (25) and (26). The RMSE of the HJB residuals for the DPI solution is 2×10^{-3}, computed on a random sample of 8,192,000 observations (2^{14} parallel simulations of size 2,000) sampled from the ergodic distribution. The value of the parameters are as follows: $\gamma=2$, $\rho=0.04$, $\mu_{\mathbf{x}}(\mathbf{x})=-0.45\mathbf{x}$, $\sigma_{\mathbf{x},i}(\mathbf{x})=0.1\mathbf{e}_i$, $\sigma_{\mathbf{r}}=0.2\mathbf{e}_{N+1}$, and $N=10$. The functions $\xi(\mathbf{x})$ and $r(\mathbf{x})$ are given by Equations (23) and (24). The network was trained in approximately 1 minute.

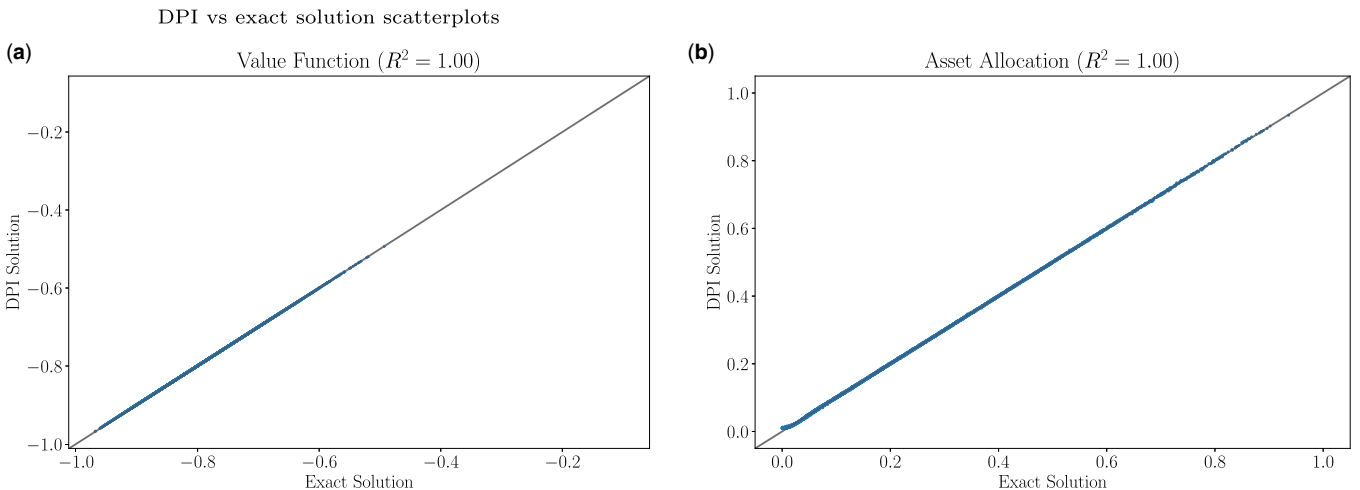

Figure 12
DPI vs exact solution scatterplots
The figure shows the scatterplots of the value function (left panel) and portfolio share (right panel) computed using the DPI method against the exact solution given by Equations (25) and (26).

Table 2
List of state variables driving the expected returns of assets

Variable	Description	Mean	S.D.(%)
π_t	log inflation	0.032	2.3
$y_t^{\$}(1)$	log 1-year nominal yield	0.043	3.1
$yspr_t^{\$}$	log 5-year minus 1-year nominal spread	0.006	0.7
Δz_t	log real GDP growth	0.030	2.4
Δd_t	log stock dividend-to-GDP growth	−0.002	6.3
d_t	log stock dividend-to-GDP level	−0.270	30.5
pd_t	log stock price-to-dividend ratio	3.537	42.6
$\Delta\tau_t$	log tax revenue-to-GDP growth	0.000	5
τ_t	log tax revenue-to-GDP level	−1.739	6.5
Δg_t	log spending-to-GDP growth	0.006	7.6
g_t	log spending-to-GDP level	−1.749	12.9

The table shows the list of 11 state variables driving the expected returns in our economy, along with their mean and standard deviation. The data are collected from https://www.publicdebtvaluation.com/data.

$N \times 1$ vector of market prices of risk $\boldsymbol{\eta}(\mathbf{x}_t)$ are assumed to be affine functions of $\mathbf{x}_t$, with $r(\mathbf{x}_t) = r_0 + \boldsymbol{r}_1^\top \mathbf{x}_t$ and the $\boldsymbol{\eta}(\mathbf{x}_t) = \boldsymbol{\eta}_0 + \boldsymbol{\eta}_1^\top \mathbf{x}_t$.

Estimation of the state dynamics. We assume that there are $N = 11$ state variables, comprised of the financial and macroeconomic variables described in Table 2. These variables include standard bond and stock market predictors as well as relevant macroeconomic variables.

Data on the state variables $\mathbf{x}_t$ are sampled in discrete intervals and their process can be estimated by fitting a VAR(1):

$$\mathbf{x}_t = \Psi \mathbf{x}_{t-1} + \mathbf{u}_t, \tag{28}$$

where Ψ is a $N \times N$ matrix of coefficients, $\mathbf{u}_t = \mathbf{B}\boldsymbol{\epsilon}_t$ is a $N \times 1$ vector of shocks, $\mathbf{B}$ is a $N \times N$ lower-triangle matrix of loadings, and $\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, I_N)$. The time-integrated version of the continuous-time process in Equation (27) implies specific values for Ψ and $\mathbf{B}$. We can then recover the continuous-time parameters Φ and $\sigma_{\mathbf{x}}$ from the discrete-time VAR by solving an inverse problem in the spirit of Campbell et al. (2004), that is, finding the continuous-time parameters that when time-integrated deliver the estimated VAR coefficients. The internet appendix describes this problem in detail.

Estimation of the state-price density. Given the assumption on the affine structure of $r(\mathbf{x})$ and $\boldsymbol{\eta}(\mathbf{x})$, we derive closed-form expressions for bond yields and expected stock returns and then search for the parameters r_0, $\boldsymbol{r}_1$, $\boldsymbol{\eta}_0$ and $\boldsymbol{\eta}_1$ to minimize the squared residuals between the model-implied time-integrated values and the corresponding data for 12 time series: 1-, 2-, 5-, 10-, 20-, and 30-year nominal yields, 5-, 7-, 10-, 20-, and 30-year real yields, and expected stock returns. Figure 13 shows the model fit for six selected series. Similar results hold for the remaining six variables. As illustrated, the model can accurately capture the evolution of nominal and real bonds of different maturities. Moreover, the equity premium implied by the model closely matches the conditional equity premium in the data implied by the VAR.

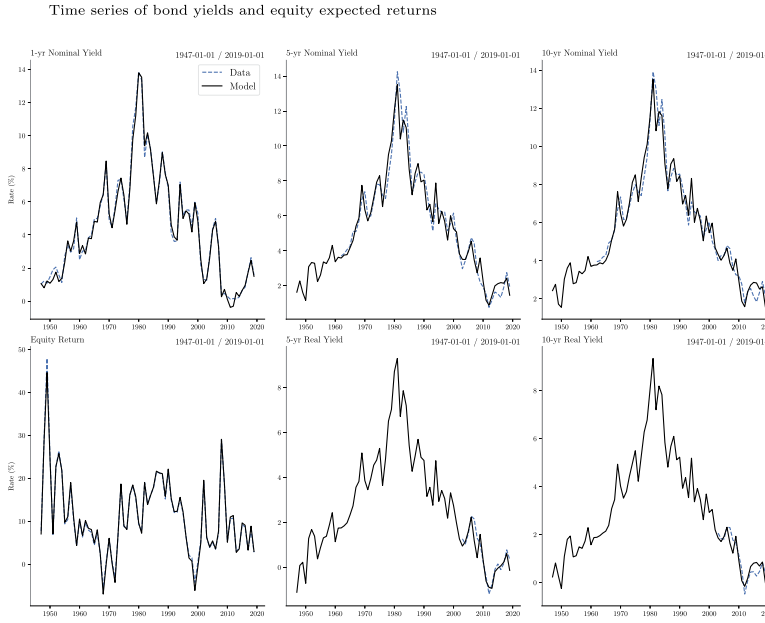

Figure 13
Time series of bond yields and equity expected returns
The figure shows the time series for nominal yields, real yields, and equity expected return for the model (solid black line) and the data (dashed blue line). The maturities for the nominal yields are 1, 5, and 10 years. The maturities for the real yields are 5 and 10 years.

Given $(r(\mathbf{x}_t), \boldsymbol{\eta}(\mathbf{x}_t))$ and the state dynamics in Equation (27), we can derive the 5×1 vector of expected excess return $\boldsymbol{\xi}(\mathbf{x}_t)$ for risky assets and the matrix of loadings on the Brownian shocks $\sigma_{\mathbf{R}} \in \mathbb{R}^{5\times 11}$ to describe the investor's investment opportunity set and the dynamics of the investor's wealth.[35]

The investor's optimization problem. With the investment opportunity set fully described, we turn to the optimization problem faced by the investor. The agent chooses consumption C_t and portfolio shares $\boldsymbol{\alpha}_t \in \mathbb{R}^{5\times 1}$ to solve the following optimization problem:

$$V(W, \mathbf{x}) = \max_{\{C_t, \boldsymbol{\alpha}_t\}_0^\infty} \mathbb{E}_0 \left[\int_0^\infty f(C_t, V_t) dt \right], \tag{29}$$

subject to the state dynamics in Equation (27), the wealth dynamics

$$dW_t = \left(W_t (r(\mathbf{x}_t) + \boldsymbol{\alpha}_t^\top \boldsymbol{\xi}(\mathbf{x}_t)) - C_t \right) dt + W_t \boldsymbol{\alpha}_t^\top \sigma_{\mathbf{R}} d\mathbf{Z}_t,$$

the position limits $0 \le \alpha_{j,t} \le 1$ for $j = 1, \ldots, 5$, the natural borrowing limit $W_t \ge 0$, and initial conditions $W_0 = W$ and $\mathbf{x}_0 = \mathbf{x}$. The preference aggregator is

[35] See the internet appendix for a detailed derivation of the model and a thorough discussion of the estimation process for this exercise.

given by $f(C,V)=\rho\frac{(1-\gamma)V}{1-\psi^{-1}}\left(\left(\frac{C}{((1-\gamma)V)^{\frac{1}{1-\gamma}}}\right)^{1-\psi^{-1}}-1\right)$, where ρ is the time-preference parameter. The position limits imply the investor is not allowed to short sell or use leverage.

Optimal allocation. We use the DPI algorithm to find the optimal consumption and portfolio plans that solve the optimization problem in Equation (29). Panel A of Figure 14 shows the evolution of expected returns for the six asset classes. Expected returns show substantial variability over our sample period and exhibit a strong cyclical component, with large spikes in expected excess returns during recessions.

Panel B of Figure 14 shows the optimal allocation for an investor with a coefficient of risk aversion $\gamma=20$ and EIS $\psi=0.5$.[36] The solution shows that the investor engages in market timing to a great extent. For instance, the investor held a substantial amount of stocks during the 1950s and 1960s, but she was nearly out of the stock market during the early 1970s. Similarly, the optimal solution was to essentially stay away from stocks during the early 2000s, at the height of the dot-com bubble.

An important feature of the optimal portfolio is the substantial demand for inflation-protected bonds. The holdings of medium- and long-term real bonds are substantial during several periods in our sample. Even though inflation-protected bonds were only introduced in the United States in the late 1990s, our model enables us to assess what would the optimal holdings of these bonds be if they were available throughout our sample period.

Figure 14 also shows a rich pattern of substitution between stocks and bonds. For instance, as investors reduce their exposure to stocks in the early 1970s, they substantially increase their holdings of long-term real bonds. In contrast, as investors reduce again their exposure to stocks during the early 2000s, they shift their portfolio mostly to nominal bonds this time. To better understand these substitution patterns, we consider next how the policy functions vary with each state variable in isolation.

Policy functions. Figure 15 shows how the consumption-wealth ratio and the portfolio share of the different assets respond to changes in the state variables. Each line shows the response of an outcome as we vary a given state variable by ±1 standard deviations, while we keep the remaining variables at their average level.

Panel A of Figure 15 shows how the investor's consumption behavior responds to changes in the state variables. An increase in expected returns, as captured, for example, by higher short-term interest rates or lower price-dividend ratio, leads to an increase in the consumption-wealth ratio. Hence, the investor saves *less* when returns are high, consistent with the income

[36] The internet appendix discusses how the preference parameters affect the optimal allocation.

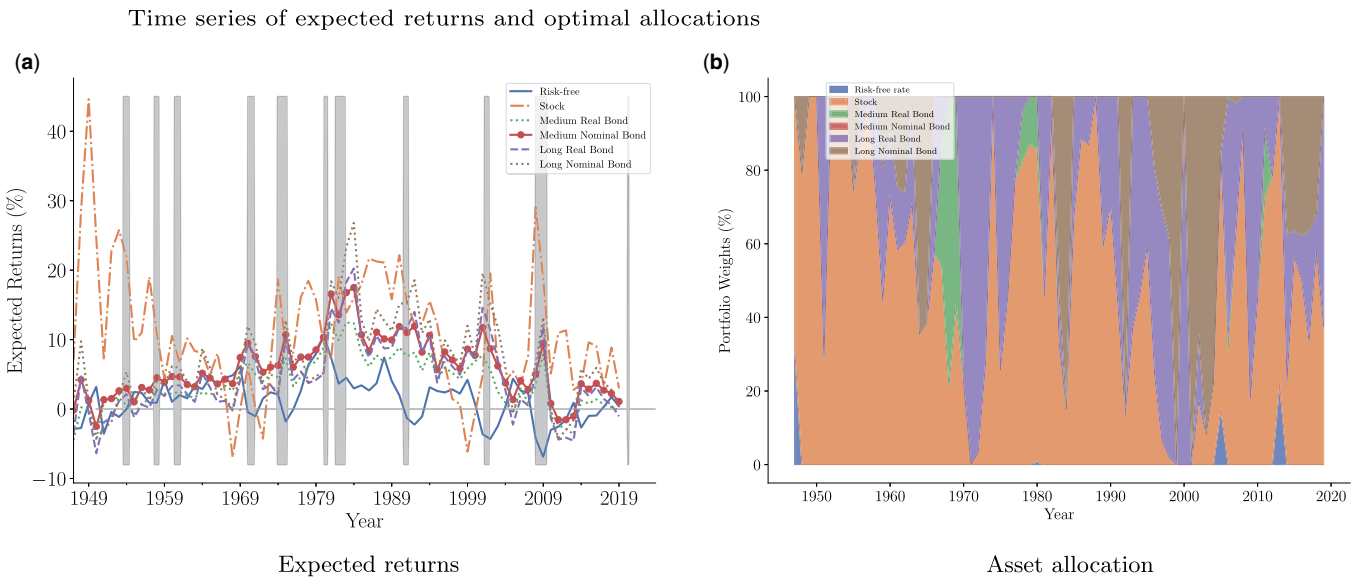

Figure 14
Time series of expected returns and optimal allocations
Panel A shows the time series of expected returns implied by the model for five asset classes: equities, nominal and real long-term bonds (i.e., 10-year maturity), and nominal and real medium-term bonds (i.e., 5-year maturity), for the period 1949–2019. The NBER recession periods are indicated as grey bars. Panel B shows the time series of the optimal asset allocation computed using the DPI algorithm for an investor with recursive utility solving the optimization problem in Equation (29), given the dynamics of expected returns shown in panel A and preferences parameters $\rho = 0.04$, $\gamma = 20$, and $\psi = 0.5$.

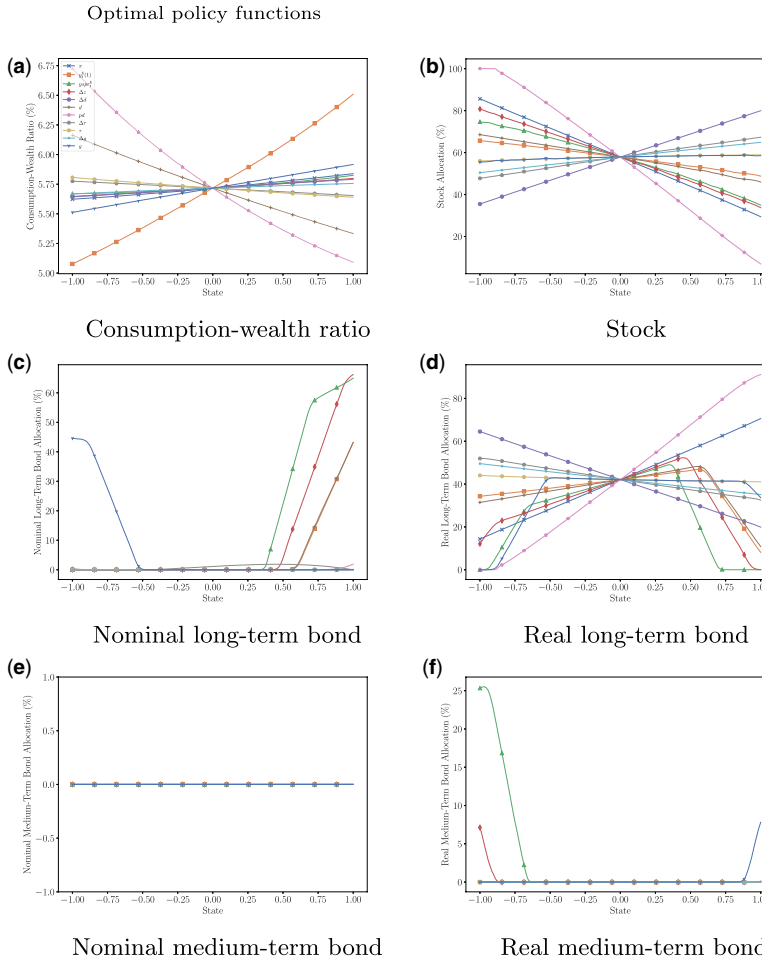

Figure 15
Optimal policy functions
The panels show the optimal policies computed with the DPI algorithm as a function of the 11 state variables. The effects on consumption-wealth ratio, stock, nominal long-term bond, real long-term bond, nominal medium-term bond, and real medium-term bond are represented in panels A, B, C, D, E, and F, respectively. Long-term bonds have 10-year maturity and medium-term bonds mature in 5 years. The x-axis is measured in standard deviations for each state variable.

effect dominating the substitution effect in savings decision, in line with our assumption of a low EIS ($\psi = 0.5$).

As expected, the portfolio share of stocks is decreasing in the price-dividend ratio, as a high price-dividend ratio forecasts lower future returns. More interestingly, panel B of Figure 15 shows that the share of stocks on the portfolio also responds to variables typically associated with the bond market, such as the yield spread or the inflation rate. This captures the substitution pattern between stocks and bonds.

The demand for long-term real bonds is naturally increasing in the inflation rate, as these bonds are designed to provide inflation protection. For small deviations of inflation from its mean, the investor obtains this protection only from long-term bonds, while for large deviations the investor uses both medium- and long-term bonds. We also find that the demand for inflation-protected bonds is very sensitive to movements in the price-dividend ratio, a standard stock market predictor. Notice this is not a mechanical effect, as the investor could have chosen instead to raise her holdings of short-term bonds or long-term nominal bonds when stocks become less attractive due to a high price-dividend ratio.

The way the investor reallocates her portfolio is more intricate for changes in the yield spread. An increase in the yield spread leads to a reduction in stock holdings and an initial increase in real long-term bonds. For larger deviations of the yield spread, the investor shifts away from real bonds toward long-term nominal bonds. This behavior leads to highly nonlinear policy functions that are unlikely to be captured by the log-linear approximations commonly used in portfolio problems. Moreover, for the range of parameters we consider, the agent does not invest in the medium-term nominal bond.

Portfolio sensitivities. In our last analysis, we investigate what are the main economic factors driving changes in portfolio allocation. To assess that, for a given asset j, we decompose the change in its weights from time t to $t+1$ as:

$$\alpha_j(\mathbf{x}_{t+1}) - \alpha_j(\mathbf{x}_t) \approx \sum_{i=1}^{11} \frac{\partial \alpha_j}{\partial x_i} \left(x_{i,t+1} - x_{i,t}\right),$$

and define the sensitivity of asset j to state variable i at time $t+1$ as:

$$s_{ij,t+1} \equiv \frac{\left| \frac{\partial \alpha_j}{\partial x_i} \left(x_{i,t+1} - x_{i,t}\right) \right|}{\sum_{\iota=1}^{11} \left| \frac{\partial \alpha_j}{\partial x_\iota} \left(x_{\iota,t+1} - x_{\iota,t}\right) \right|}. \tag{30}$$

By construction, the sum of the sensitivities of an asset allocation with respect to the 11 state variables adds up to one, which allows us to interpret this measure as the relative importance of each state variable for a given asset allocation, at a given time.

Table 3 shows the sensitivities for all assets averaged over our sample period. Movements in the price-dividend ratio account on average for 22% of the variability in the share invested on stocks, while the yield spread accounts for 14%, inflation 11%, and fiscal variables 15%. Interestingly, all fiscal variables together account for more than 20% of the variability in real and nominal long-term bonds. This is more than the fraction explained by the short-rate or the term spread, commonly used predictors of bond returns.

Table 3
Sensitivities

	π	$y_t^{\$}(1)$	$yspr_t^{\$}$	Δz	Δd	d	pd	fiscal
10y nominal	5.9	6.4	19.6	17.3	11.6	3.1	12.1	24.1
10y real	6.6	5.2	18.1	18.9	15.2	2.5	12.3	21.2
Risk-free	5.6	6.1	21.0	17.3	10.4	3.7	10.3	25.6
5y nominal	5.1	6.7	21.5	18.0	9.4	3.1	9.9	26.3
5y real	4.3	5.9	20.9	18.2	10.1	3.0	11.3	26.3
Stock	10.7	2.9	13.6	18.5	14.7	2.5	21.6	15.5

The table shows the average sensitivity of the asset allocations as a percentage of wealth with respect to each of the 11 state variables listed in Table 2. The sensitivity of the allocations is computed as in Equation (30). The column "fiscal" shows the sum of the sensitivities for all fiscal variables.

Taken together, these results indicate that the optimal portfolio follows a rich pattern that cannot be easily captured by a rule of thumb, such as a $60-40$ allocation or simple age-dependent rules. It is important to take into account market conditions as captured by key macroeconomic and financial variables.

4. Conclusion

This paper proposes a novel numerical method that alleviates the three curses of dimensionality. The method rests on three pillars. First, the method uses deep learning to represent value and policy functions. Second, the method combines Ito's lemma and automatic differentiation to compute exact expectations with negligible additional computational cost. Third, the method uses a gradient-based version of policy iteration that dispenses root-finding methods to find the optimal control for a given state. We show that the DPI method has broad applicability in several areas of finance, such as asset pricing, corporate finance, and portfolio choice, and that it can solve complex large-dimensional problems with highly nonlinear dynamical systems.

The ability to solve rich high-dimensional problems can be an invaluable tool in economic analysis. We oftentimes are forced to make assumptions that have no clear economic interest but are necessary for the model solution to be feasible. This often makes it difficult to determine whether results are due to these auxiliary assumptions or to the economically motivated ones. By significantly expanding the set of models that researchers can solve, or even potentially estimate, our methods enable researchers to focus on models that better capture the rich phenomena that we observe in modern economies, instead of focusing on models that current numerical methods can solve.

Code Availability

The replication code is available in the Harvard Dataverse at https://doi.org/10.7910/DVN/51YSDW.

A. Proofs

Proof of Proposition 1 Since the second derivative of $F(t)$ is given by

$$F''(t)=\sum_{i=1}^{m}\frac{1}{m}\mathbf{f}^{\mathbf{T}}\nabla_s V\left(\mathbf{s}+\frac{t^2}{2m}\mathbf{f}+\frac{t}{\sqrt{2}}\mathbf{g}_i\right)$$
$$+\sum_{i=1}^{m}\left(\frac{t}{m}\mathbf{f}+\frac{1}{\sqrt{2}}\mathbf{g}_i\right)^{\top}\mathbf{H}_{\mathbf{s}}V\left(\mathbf{s}+\frac{t^2}{2m}\mathbf{f}+\frac{t}{\sqrt{2}}\mathbf{g}_i\right)\left(\frac{t}{m}\mathbf{f}+\frac{1}{\sqrt{2}}\mathbf{g}_i\right),$$

evaluating it at $t=0$ gives

$$F''(0)=\nabla_{\mathbf{s}}V(\mathbf{s})^{\top}\mathbf{f}+\frac{1}{2}\mathrm{Tr}\left[\mathbf{g}^{\top}\mathbf{H}_{\mathbf{s}}V(\mathbf{s})\mathbf{g}\right]$$
$$=\frac{\mathbb{E}dV}{dt}(\mathbf{s}),$$

which concludes the proof. ■

B. Time-Varying Disasters and Epstein-Zin Preferences

In this section, we extend the method presented in Section 2 to solve an equilibrium problem where agents have Epstein-Zin preferences and the state variable is driven by a jump-diffusion process. To achieve this goal, we consider the model of Wachter (2013), which has the two aforementioned features and allows the equilibrium quantities to be characterized in closed form. Similar to the analysis of the Lucas orchard economy in Section 3.1, we use the analytical expressions to assess the accuracy of our numerical solution.

B.1 Model environment.

The economy of Wachter (2013) can be briefly summarized as follows. Aggregate dividends follow the jump-diffusion process of the form

$$\frac{dD_t}{D_{t-}}=\mu dt+\sigma dB_t+(e^{J_t}-1)dN_t,$$

where J_t is a random variable with time-invariant distribution ν, and N_t is a Poisson process with time-varying intensity process λ_t satisfying a standard Cox–Ingersoll–Ross process

$$d\lambda_t=\kappa(\overline{\lambda}-\lambda_t)dt+\sigma_\lambda\sqrt{\lambda_t}dB_{\lambda,t}.$$

All random variables are assumed to be independent. The representative investor has the continuous-time analog of Epstein-Zin preferences with unit elasticity of intertemporal substitution (EIS); that is, the value function V_t satisfies

$$V_t=\mathbb{E}_t\int_t^{\infty}f(C_s,V_s)ds,$$

where $f(C,V)=\beta(1-\gamma)V(\log C-\frac{1}{1-\gamma}\log((1-\gamma)V))$.

In this economy, the state variables driving the equilibrium quantities are the agent's wealth W_t and the time-varying intensity process λ_t. As shown in Wachter (2013), the investor's HJB equation is given by

$$0=HJB\equiv f(C,V)+V_W W\mu+V_\lambda\kappa(\overline{\lambda}-\lambda)+\frac{1}{2}V_{WW}W^2\sigma^2+\frac{1}{2}V_{\lambda\lambda}\sigma_\lambda^2\lambda$$
$$+\lambda\mathbb{E}\left[V(We^J,\lambda)-V(W,\lambda)\right]. \tag{B.1}$$

Here, $C=\beta W$, and the value function assumes the form

$$V(W_t,\lambda_t)=\frac{W_t^{1-\gamma}}{1-\gamma}I(\lambda_t), \tag{B.2}$$

where $I(\lambda_t)=e^{a+b\lambda_t}$, with a and b as coefficients given in Wachter (2013).

B.2 DPI Method with Jumps.

In the absence of jumps, the HJB equation in Equation (B.1) contains no integral and depends only on the partial derivatives of the value function, which can be easily evaluated using the methods described in Section 2.1. In the presence of jumps, however, the HJB equation in Equation (B.1) contains an integral, which in principle would require a numerical integration method. Computing this integral can be potentially very costly, making the numerical solution of models with jumps particularly challenging.[37] However, by using simulation methods analogous to the Least-Squares Monte Carlo method (LSMC hereafter) of Longstaff and Schwartz (2001), commonly used to price American options, we can bypass the evaluation of the integral.

To understand how this variation of the DPI method works, consider the following rewrite of the HJB in Equation (B.1):

$$HJB\equiv f(C,V)+\frac{\mathbb{E}^B[dV]}{dt}+\frac{\mathbb{E}^J[dV]}{dt}, \tag{B.3}$$

where

$$\frac{\mathbb{E}^B[dV]}{dt}=V_W W\mu+V_\lambda\kappa(\bar{\lambda}-\lambda)+\frac{1}{2}V_{WW}W^2\sigma^2+\frac{1}{2}V_{\lambda\lambda}\sigma_\lambda^2\lambda, \tag{B.4}$$

$$\frac{\mathbb{E}^J[dV]}{dt}=\lambda\mathbb{E}\left[V(We^J,\lambda)-V(W,\lambda)\right]. \tag{B.5}$$

The term in Equation (B.4) comes from the Brownian shock and can be computed exactly using Proposition 1, as the previous examples in the paper illustrate. The term in Equation (B.5) comes from the jump shock and involves an integral that in principle must be approximated, which can be computationally costly.

To bypass numerical integration, we simply need to implement two modifications that are surprisingly straightforward, but conceptually powerful: (i) for a given mini-batch of I samples of the state variable $\{\lambda_i\}_{i=1}^I$, approximate $\lambda\mathbb{E}\left[V(We^J,\lambda)-V(W,\lambda)\right]$ by a single random realization $\lambda_i\left(V(W_ie^{J_i},\lambda_i)-V(W_i,\lambda_i)\right)$, and (ii) use the MSE as the loss function in the policy evaluation step.

The reason these two seemingly straightforward modifications work is as follows. When using the Policy Evaluation 1 rule in Equation (17), the HJB residuals are used to construct the continuous-time Bellman target for the regression, as in Equation (15). For a given realization J_i, the regression target in Equation (15) becomes

$$V(W_i,\lambda_i)+\left(f(C_i,V(W_i,\lambda_i))+\frac{\mathbb{E}^B[dV]_i}{dt}+\lambda_i\left(V(W_ie^{J_i},\lambda_i)-V(W_i,\lambda_i)\right)\right)\Delta t, \tag{B.6}$$

where

$$\frac{\mathbb{E}^B[dV]_i}{dt}\equiv V_W W_i\mu+V_\lambda\kappa(\bar{\lambda}-\lambda_i)+\frac{1}{2}V_{WW}W_i^2\sigma^2+\frac{1}{2}V_{\lambda\lambda}\sigma_\lambda^2\lambda_i.$$

[37] See Fernández-Villaverde and Levintal (2018) for a discussion of the challenges of solving models with rare disasters.

However, as it is well known in the statistics and econometrics literature (Angrist and Pischke 2008), when the MSE is used as the loss function in the regression, minimizing this loss leads to the estimation of the *conditional expectation function*. Longstaff and Schwartz (2001) leverage this fundamental statistical result to estimate conditional expectations using regressions, and this is precisely what we do here too. Indeed, the minimization of the mean square errors using samples as in Equation (B.6) produces the conditional expectation

$$\mathbb{E}\left[V(W_i,\lambda_i)+\left(f(C_i,V(W_i,\lambda_i))+\frac{\mathbb{E}^B[dV]_i}{dt}+\lambda_i\left(V(W_i e^{J_i},\lambda_i)-V(W_i,\lambda_i)\right)\Delta t\right)\middle| W_i,\lambda_i\right]$$

$$= V(W_i,\lambda_i)+\left(f(C_i,V(W_i,\lambda_i))+\frac{\mathbb{E}^B[dV]_i}{dt}+\frac{\mathbb{E}^J[dV]_i}{dt}\right)\Delta t, \tag{B.7}$$

which is identical to the targets we would have used if we could compute the expectation $\frac{\mathbb{E}^J[dV]_i}{dt}$ exactly.

The implementation of the policy evaluation in the presence of jumps is summarized in the following pseudo-algorithm:

Algorithm 1 Policy evaluation in the presence of jumps

1: **procedure** PolicyEvaluation($\boldsymbol{\theta}_V^{j-1}$) ▷ Update the value function.
2: Draw $\{\lambda_1,\ldots,\lambda_I\}$ random points from the state space.
3: Compute $\frac{\mathbb{E}^B[dV]_i}{dt}$ using Proposition 1 as before.
4: Sample one realization of J_i per sample point.
5: Construct the vector of targets Y_i^j for the points λ_i, with $i=1,2,\ldots,I$:

$$Y_i^j \equiv V_i^{j-1}+\left(f(C_i^j,V_i^{j-1})+\frac{\mathbb{E}^B[dV]_i}{dt}+\lambda_i\left(V^{j-1}(W_i e^{J_i},\lambda_i)-V_i^{j-1}\right)\right)\Delta t.$$

6: Construct the vector of residuals as $e_i^j = V_i^{j-1}-Y_i^j$.
7: Use the SGD algorithm to update $\boldsymbol{\theta}_V^j$:

$$\boldsymbol{\theta}_V^j \leftarrow \boldsymbol{\theta}_V^{j-1}-\eta_V\frac{1}{I}\sum_{i=1}^{I} e_i^j \nabla_{\boldsymbol{\theta}_V} V_i^{j-1}.$$

8: **Return** $\boldsymbol{\theta}_V^j$ ▷ New neural network representation of V.

B.3 Numerical Solution.

Figure B.1 shows the analytical solution (dashed black line) for the value-function shifter $I(\lambda_t)$, and the numerical solution produced by the DPI method (solid blue line). As illustrated, the numerical solution is virtually indistinguishable from the analytical solution. The log RMSE of the HJB residuals is -5, demonstrating that the DPI method is able to provide an accurate solution to this asset pricing problem in a much more complex environment with time-varying disaster risk and recursive preferences.

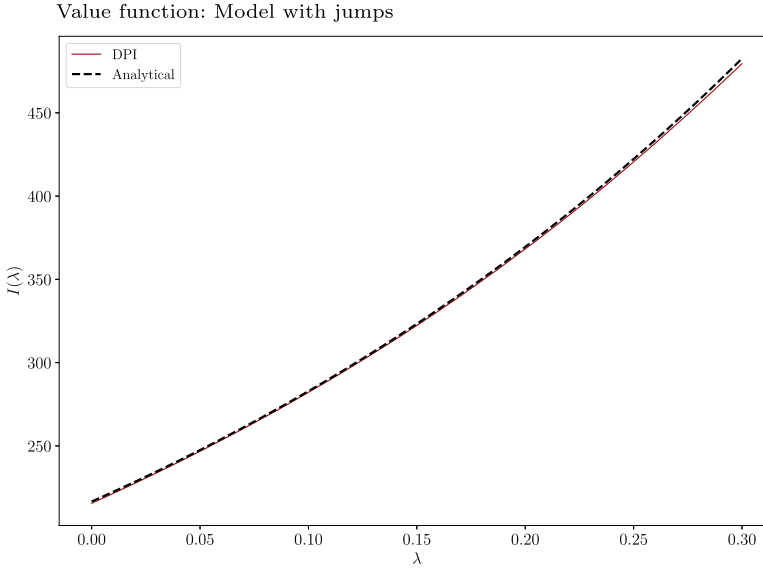

Figure B.1
Value function: Model with jumps
The figure shows the value-function shifter $I(\lambda)$ for the solution using the DPI method (red solid line) and the exact solution (black dashed line). Parameter values are as in Wachter (2013). For the network architecture, we use LayerNormMLP with SILU activation with [32, 32] hidden units. Each iteration is performed on a random batch of size 4,096. The optimizer is Adam with default parameters (learning rate = 10^{-3}, $\beta_1 = 0.9$, and $\beta_2 = 0.999$).

References

Achdou, Y., F. J. Buera, J.-M. Lasry, P.-L. Lions, and B. Moll. 2014. Partial differential equation models in macroeconomics. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 372.

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

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

Andrews, I., M. Gentzkow, and J. M. Shapiro. 2017. Measuring the sensitivity of parameter estimates to estimation moments. *Quarterly Journal of Economics* 132:1553–92.

———. 2020. Transparency in structural research. *Journal of Business & Economic Statistics* 38:711–22.

Angrist, J. D., and J.-S. Pischke. 2008. *Mostly harmless econometrics: An empiricist's companion*. Princeton University Press.

Armstrong, T. B., and M. Koles'ar. 2021. Sensitivity analysis using approximate moment condition models. *Quantitative Economics* 12:77–108.

Azinovic, M., L. Gaegauf, and S. Scheidegger. 2022. Deep equilibrium nets. *International Economic Review* 63:1471–525.

Baird, L. 1995. Residual algorithms: Reinforcement learning with function approximation. In A. Prieditis and S. Russell, eds., *Machine Learning Proceedings 1995*, 30–7. San Francisco CA: Morgan Kaufmann.

Bali, T. G., H. Beckmeyer, M. Mörke, and F. Weigert. 2023. Option return predictability with machine learning and big data. *Review of Financial Studies* 36:3548–602.

Baydin, A. G., B. A. Pearlmutter, and A. A. Radul. 2015. Automatic differentiation in machine learning: A survey. *CoRR* abs/1502.05767.

Bellman, R. 1957. *Dynamic programming*. 1st ed. Princeton, NJ, USA: Princeton University Press.

Bianchi, D., M. Buüchner, and A. Tamoni. 2021. Bond risk premiums with machine learning. *Review of Financial Studies* 34:1046–89.

Bretscher, L., J. Fern'andez-Villaverde, and S. Scheidegger. 2022. Ricardian business cycles. Working Paper, Swiss Finance Institute.

Brumm, J., C. Krause, A. Schaab, and S. Scheidegger. 2022. *Sparse Grids for Dynamic Economic Models*. In: Oxford Research Encyclopedia of Economics and Finance.

Brumm, J., and S. Scheidegger. 2017. Using adaptive sparse grids to solve high- dimensional dynamic models. *Econometrica* 85:1575–612.

Brunnermeier, M., and Y. Sannikov. 2016. Macro, money, and finance: A continuous-time approach. vol. 2 of *Handbook of Macroeconomics*, 1497–545. Elsevier.

Brunnermeier, M. K., and Y. Sannikov. 2014. A macroeconomic model with a financial sector. *American Economic Review* 104:379–421.

Bybee, L., B. T. Kelly, A. Manela, and D. Xiu. 2021. Business news and business cycles. *Working Paper, Yale University*.

Campbell, J. Y., G. Chacko, J. Rodriguez, and L. M. Viceira. 2004. Strategic asset allocation in a continuous-time var model. *Journal of Economic Dynamics and Control* 28:2195–214.

Campbell, J. Y., and L. M. Viceira. 1999. Consumption and portfolio decisions when expected returns are time varying. *Quarterly Journal of Economics* 114:433–95.

Cao, S., W. Jiang, B. Yang, and A. L. Zhang. 2023. How to talk When a machine is listening: Corporate disclosure in the age of AI. *Review of Financial Studies* 36:3603–42.

Catherine, S., M. Ebrahimian, D. Sraer, and D. Thesmar. 2022. Robustness checks in structural analysis. Working Paper, University of Pennsylvania.

Cauchy, A. 1847. M'ethode g'en'erale pour la r'esolution des systemes d"equations simultan'ees. *Comptes rendus de l'Acad'emie des Sciences* 25:536–8.

Chen, H., A. Didisheim, and S. Scheidegger. 2021. Deep structural estimation: With an application to option pricing. *arXiv, preprint, https://arxiv.org/abs/2102.09209*.

Chen, L., M. Pelger, and J. Zhu. 2023. Deep learning in asset pricing. *Management Science*.

Cochrane, J. H. 1991. Production-based asset pricing and the link between stock returns and economic fluctuations. *Journal of Finance* 46:209–37.

Cochrane, J. H., F. A. Longstaff, and P. Santa-Clara. 2008. Two trees. *Review of Financial Studies* 21:347–85.

Crandall, M. G. 1995. Viscosity solutions: A primer. In *Viscosity Solutions and Applications*.

Cybenko, G. 1989. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signals and Systems* 2:303–14.

Daniel, K., and S. Titman. 2006. Market reactions to tangible and intangible information. *Journal of Finance* 61:1605–43.

Drechsler, I., A. Savov, and P. Schnabl. 2018. A model of monetary policy and risk premia. *Journal of Finance* 73:317–73.

Duarte, V. 2018. Gradient-based structural estimation. Working Paper, University of Illinois at Urbana-Champaign.

Duarte, V., D. Duarte, J. Fonseca, and A. Montecinos. 2020. Benchmarking machine-learning software and hardware for quantitative economics. *Journal of Economic Dynamics and Control* 111:103796–.

Duarte, V., J. Fonseca, A. S. Goodman, and J. A. Parker. 2021. Simple allocation rules and optimal portfolio choice over the lifecycle. Working Paper, University of Illinois at Urbana-Champaign.

Epperson, J. F. 1987. On the runge example. *The American Mathematical Monthly* 94:329–41.

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

Fernandez-Villaverde, J., and O. Levintal. 2018. Solution methods for models with rare disasters. *Quantitative Economics* 9:903–44.

Folini, D., A. Friedl, F. Kuübler, and S. Scheidegger. 2024. The Climate in Climate Economics*. *The Review of Economic Studies* rdae011.

Fuster, A., P. Goldsmith-Pinkham, T. Ramadorai, and A. Walther. 2022. Predictably unequal? the effects of machine learning on credit markets. *Journal of Finance* 77:5–47.

Gârleanu, N., and L. H. Pedersen. 2013. Dynamic trading with predictable returns and transaction costs. *Journal of Finance* 68:2309–40.

Goodfellow, I., Y. Bengio, and A. Courville. 2016. *Deep learning*. MIT Press.

Gopalakrishna, G. 2021. Aliens and continuous time economies. Working Paper 21-34, Swiss Finance Institute.

Griewank, A., and A. Walther. 2008. *Evaluating derivatives: Principles and techniques of algorithmic differentiation*. 2nd ed. Princeton, NJ: Society for Industrial and Applied Mathematics.

Gu, S., B. Kelly, and D. Xiu. 2020. Empirical asset pricing via machine learning. *Review of Financial Studies* 33:2223–73.

Han, J., Y. Yang, and W. E. 2022. Deepham: A global solution method for heterogeneous agent models with aggregate shocks. *arXiv, preprint, https://arxiv.org/abs/2112.14377*.

Haugh, M. B., and L. Kogan. 2004. Pricing american options: A duality approach. *Operations Research* 52:258–70.

Heess, N., D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver. 2017. Emergence of locomotion behaviours in rich environments. *CoRR* abs/1707.02286.

Hennessy, C. A., and T. M. Whited. 2007. How costly is external financing? evidence from a structural estimation. *Journal of Finance* 62:1705–45.

Hornik, K. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4:251–7.

Howard, R. A. 1960. *Dynamic programming and markov processes*. Cambridge, MA: MIT Press.

Jarrett, K., K. Kavukcuoglu, and Y. LeCun. 2009. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, 2146–53. IEEE.

Jiang, Z., H. Lustig, S. Van Nieuwerburgh, and M. Z. Xiaolan. 2019. The us public debt valuation puzzle. Working Paper, Northwestern University.

Judd, K. L., L. Maliar, S. Maliar, and R. Valero. 2014. Smolyak method for solving dynamic economic models: Lagrange interpolation, anisotropic grid and adaptive domain. *Journal of Economic Dynamics and Control* 44:92–123.

Kargar, M. 2021. Heterogeneous intermediary asset pricing. *Journal of Financial Economics* 141:505–32.

Kase, H., L. Melosi, and M. Rottner. 2022. Estimating nonlinear heterogeneous agents models with neural networks. *Discussion Paper, CEPR*.

Koijen, R. S., and S. Van Nieuwerburgh. 2011. Predictability of returns and cash flows. *Annual Review of Financial Economics* 3:467–91.

Krizhevsky, A., I. Sutskever, and G. E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., *Advances in Neural Information Processing Systems 25*, 1097–105. Curran Associates, Inc.

Ledoit, O., and M. Wolf. 2004. Honey, I shrunk the sample covariance matrix. *Journal of Portfolio Management* 30:110–.

Lettau, M., and S. Ludvigson. 2001. Consumption, aggregate wealth, and expected stock returns. *Journal of Finance* 56:815–49.

Lewellen, J. 2015. The cross-section of expected stock returns. *Critical Finance Review* 4:1–44.

Li, K., F. Mai, R. Shen, and X. Yan. 2021. Measuring corporate culture using machine learning. *Review of Financial Studies* 34:3265–315.

Liaw, R., E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. 2018. Tune: A research platform for distributed model selection and training. *arXiv, preprint, https://arxiv.org/abs/1807.05118*.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971.

Ljungqvist, L., and T. Sargent. 2000. *Recursive macroeconomic theory*. Cambridge: MIT Press.

Longstaff, F. A., and E. S. Schwartz. 2001. Valuing American options by simulation: A simple least-squares approach. *Review of Financial Studies* 14:113–47.

Lucas, R. E. 1978. Asset prices in an exchange economy. *Econometrica* 46:1429–45.

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

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

Martin, I. 2013. The Lucas orchard. *Econometrica* 81:55–111.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518:529–33.

Moreira, A., and A. Savov. 2017. The macroeconomics of shadow banking. *Journal of Finance* 72:2381–432.

Nagel, S. 2021. *Machine learning in asset pricing*, vol. 1. Princeton University Press.

Norets, A. 2012. Estimation of dynamic discrete choice models using artificial neural network approximations. *Econometric Reviews* 31:84–106.

Parra-Alvarez, J. C. 2018. A comparison of numerical methods for the solution of continuous-time dsge models. *Macroeconomic Dynamics* 22:1555–83.

Pedersen, L. H., A. Babu, and A. Levine. 2021. Enhanced portfolio optimization. *Financial Analysts Journal* 77:124–51.

Powell, W. B. 2007. *Approximate dynamic programming: Solving the curses of dimensionality (wiley series in probability and statistics)*. New York: Wiley-Interscience.

Rapin, J., and O. Teytaud. 2018. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad.

Ross, S. A. 1976. Options and efficiency. *Quarterly Journal of Economics* 90:75–89.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1988. Neurocomputing: Foundations of research. chap. Learning Representations by Back-propagating Errors, 696–9. Cambridge: MIT Press.

Sadhwani, A., K. Giesecke, and J. Sirignano. 2021. Deep learning for mortgage risk. *Journal of Financial Econometrics* 19:313–68.

Sauzet, M. 2021. Projection methods via neural networks for continuous-time models. Working Paper, Boston University.

Schaul, T., D. Horgan, K. Gregor, and D. Silver. 2015. Universal value function approximators. In *International conference on machine learning*, 1312–20. PMLR.

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

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529:484–9.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550:354 EP–.

Song, X., S. Perel, C. Lee, G. Kochanski, and D. Golovin. 2023. Open source vizier: Distributed infrastructure and API for reliable and flexible blackbox optimization.

Stokey, N., R. Lucas, and E. Prescott. 1989. *Recursive methods in economic dynamics*. Cambridge: Harvard University Press.

Sutton, R. S., and A. G. Barto. 1998. *Introduction to reinforcement learning*. 1st ed. Cambridge, MA, USA: MIT Press.

Wachter, J. A. 2013. Can time-varying risk of rare disasters explain aggregate stock market volatility? *Journal of Finance* 68:987–1035.

Welch, I., and A. Goyal. 2008. A comprehensive look at the empirical performance of equity premium prediction. *Review of Financial Studies* 21:1455–508.