UNIL | Université de Lausanne
Département de finance

EPFL

FINANCE RESEARCH SEMINAR SUPPORTED BY UNIGESTION

MACHINE LEARNING FOR CONTINUOUS-TIME FINANCE

Victor Duarte,
Gies College of Business UIUC, Champaign Illinois

This paper proposes an algorithm for solving a large class of nonlinear continuous-time models in finance and economics. First, I recast the problem of solving the corresponding nonlinear partial differential equations as a sequence of supervised learning problems. Second, I prove that the computational cost of evaluating the exact continuous-time Bellman residuals does not increase in the number of state variables, allowing for the solution of high-dimensional problems. To illustrate the method, I solve canonical asset pricing models featuring recursive preferences, endogenous labor supply, irreversible investment, and state spaces with up to ten dimensions.

Friday, February 14th, 2020, 10:30-12:00
Room 126, Extranef building at the University of Lausanne

UNIGESTION

MACHINE LEARNING FOR CONTINUOUS-TIME FINANCE

Victor Duarte
Federal Reserve Bank of Dallas
victor.duarte@dal.frb.org *

September 24, 2018

ABSTRACT

This paper proposes an algorithm for solving a large class of nonlinear continuous-time models in finance and economics. First, I recast the problem of solving the corresponding nonlinear partial differential equations as a sequence of *supervised learning* problems. Second, I prove that the computational cost of evaluating the exact continuous-time Bellman residuals does not increase in the number of state variables, allowing for the solution of high-dimensional problems. To illustrate the method, I solve canonical asset pricing models featuring recursive preferences, endogenous labor supply, irreversible investment, and state spaces with up to ten dimensions.

1 Introduction

Dynamic programming is one of the cornerstones of modern financial economics. Investors, managers, households, and governments all choose their actions to maximize their respective value functions. Furthermore, in the absence of arbitrage opportunities, stock prices are themselves value functions under the risk-neutral probability.

Dynamic programming is, however, plagued by the "curse of dimensionality" (Bellman (1957))—it becomes exponentially more challenging in terms of computing time and memory as the number of state variables increases. The curse of dimensionality is actually made up of three curses (Powell (2007)). The first curse is that of approximating a high-dimensional, nonlinear function. The second curse is the computation of expectations involved in Bellman iterations. Last, the third curse corresponds to the choice of optimal policies. Each of these challenges imposes severe limitations on the advancement of finance theory. Therefore, most financial economics research today is restricted to models featuring either small state spaces or linearized solutions.[2] While linearized solutions often yield good approximations for certain dynamic stochastic models in macroeconomics, the accuracy and applicability is more questionable in financial economics applications. Pohl et al. (2018), for instance, show that the ubiquitous Campbell and Shiller (1988) log linearization generates errors as high as 70% in long-run risk asset pricing models, highlighting the need for global solution methods in financial economics.

This paper proposes a novel algorithm that handles nonlinear stochastic dynamic programming problems with large state spaces, opening the possibility of the study of richer models. To address the first curse of dimensionality, I incorporate recent and promising developments in the field of reinforcement learning[3]. Specifically, I use deep neural networks to represent value functions and optimal policies. The main theoretical result in this paper is showing how to overcome

*This paper benefited from comments by Markus Brunnermeier, Diogo Duarte, Julia Fonseca, Daniel Greenwald, Leonid Kogan, Deborah Lucas, Karel Mertens, Alexis Montecinos, Jonathan Parker, Alex Richter, Dejanir Silva, Adrien Verdelhan, Gianluca Violante, and seminar participants at the WEAI Annual Meeting, the Macro Financial Modeling Summer Session, and the MIT Finance Seminar. Generous financial support for this project was provided by The Becker Friedman Institute's Macro Financial Modeling Initiative.

[2] I will call a state space *small* if it has less than five dimensions and *large* otherwise. Canonical papers featuring small state spaces are Campbell and Cochrane (1999), Campbell and Viceira (1999), Bansal and Yaron (2004), Hennessy and Whited (2005), Barro (2006), Sannikov (2008), Kogan and Papanikolaou (2013), He and Krishnamurthy (2013), and Brunnermeier and Sannikov (2014), to name a few.

[3] Reinforcement learning is the subfield in machine learning that studies intertemporal optimization. See Sutton and Barto (1998), Powell (2007) and Bertsekas and Tsitsiklis (1996).

the second curse of dimensionality—I prove that, in continuous time, the computational complexity of the Bellman updates does not depend on the number of state variables and scales linearly in the number of shocks. Last, to address the third curse, I employ a type of generalized policy iteration (Sutton and Barto (1998)) based on policy gradients, similar to Lillicrap et al. (2015).

Obviously, the use of neural networks to approximate value functions is not new. In the optimal control literature, for example, *shallow* neural networks[4] have been used in small-scale, deterministic applications (Coulom (2004), Abu-Khalaf and Lewis (2005), Tassa and Erez (2007), and Vamvoudakis and Lewis (2010)). Shallow networks have also been used in option pricing (Haugh and Kogan (2004)) and for the estimation of discrete choice models (Norets (2012)). All these papers belong to the era that preceded the *deep learning* revolution that started in October 2012 (Krizhevsky et al. (2012)).

For our purposes, we can think of deep learning as a technology to represent high-dimensional, nonlinear functions, using neural networks with multiple hidden layers. The change from one to multiple layers poses many challenges but also offers many opportunities. The biggest advantage of deep learning over older machine learning models is its unparalleled ability to represent very high-dimensional functions (Montufar et al. (2014) and Mhaskar et al. (2016)). Unfortunately, training deep learning models is harder. For instance, standard second-order optimization methods (Levenberg (1944), Marquardt (1963), Byrd et al. (1995)) do not perform well with deep neural networks, so a new generation of first-order stochastic optimization algorithms were developed to train deep learning models (Zeiler (2012), McMahan et al. (2013), Kingma and Ba (2014)). For good performance, it is also important to initialize the network parameters carefully (Glorot and Bengio (2010)) and use some type of automatic normalization (Ioffe and Szegedy (2015), Lei Ba et al. (2016)). Finally, in practice, finite differences or symbolic differentiation cannot be used to compute the gradients of a deep neural network with respect to the network parameters. One has to resort to *backpropagation* (Rumelhart et al. (1988)), which is both faster and more precise than the aforementioned alternatives. Until very recently, these obstacles were an insurmountable entry barrier for most economists, and that is why, so far, these technologies have not been used for solving economic models.

Fortunately, the entry barrier to deep learning has decreased dramatically over the last couple of years. Indeed, tech companies such as Google, Microsoft, Facebook, and Amazon have open-sourced their deep learning software, making deep learning accessible for anyone with basic programming skills.

In a groundbreaking article, (Mnih et al. (2015)) combined deep learning with reinforcement learning to solve nontrivial intertemporal optimization problems with hundreds of state variables, founding the striving field of *deep reinforcement learning*. Since then, deep neural networks have been successfully employed to solve dynamic stochastic problems, such as mastering the complex game of Go (Silver et al. (2016)) and advanced robotic control (Heess et al. (2017)). In December 2017, a deep reinforcement learning algorithm (Silver et al. (2017)) bested the highest-rated chess engine, after a mere four hours of training.

This paper is not, however, a direct application of these successful techniques. The crucial difference is how I compute the expectations in the Bellman target. In continuous time, I show that Ito's Lemma can be combined with backpropagation to compute exact Bellman residuals with computation complexity that does not depend on the number of state variables.[5] This is a counterintuitive result and interesting by itself. Ito's Lemma involves the computation of the Hessian matrix of the value function, and therefore one would expect that the computational cost of applying Ito's Lemma would increase at least quadratically with the number of states. That is the case of the finite-differences methods proposed in Achdou et al. (2014) and Brunnermeier and Sannikov (2016), and that is why it would be hard to scale up these methods to large state spaces. I show, however, how to compute exact drifts and volatilities with Ito's Lemma without the computation of the Hessian matrix.

To illustrate the scope and applicability of the algorithm, I solve canonical financial economics models featuring recursive preferences, endogenous labor supply, irreversible investment, constrained optimization, and state spaces with up to ten dimensions. Furthermore, I propose a setting to test and evaluate arbitrary solution methods. In the context of a neoclassical growth model (NGM), given *any* value function, the productivity function is reverse engineered so that the Hamilton-Jacobi-Bellman (HJB) equation corresponding to the optimization problem is identically zero. We will call this procedure the *NGM lab*. It provides a testing ground for solution methods and an objective way of comparing them.

The paper is supplemented by a library built on top of TensorFlow[6], the open-source machine learning library created and maintained by Google, that does most of the heavy lifting, such as applying Ito's Lemma to arbitrary multivariate functions, creating deep neural network representations of value functions, and implementing the Bellman iteration

[4] Shallow networks are neural networks with a single hidden layer. Section 2 defines neural networks and hidden layers.

[5] Throughout the paper, whenever I say "exact" I mean exact up to machine precision.

[6] See https://www.tensorflow.org/.

proposed in the paper. The user is not required to supply any equation, with the exception of the model setup. To the best of my knowledge, this is only solution method for large state space models in economics that requires no paper and pencil computations, including Euler equations and other first-order conditions. Furthermore, all models in this paper were solved with fewer than 50 lines of code.

Finally, it is important to highlight that the method advanced in this paper does not necessarily dominate any other solution method, like projection (Judd (1992)), perturbation (Judd and Guu (1997)), finite differences (Achdou et al. (2014) and Brunnermeier and Sannikov (2016)), Smolyak-based methods (Krueger and Kubler (2004), Winschel and Kratzig (2010) and Judd et al. (2014)), sparse-grid methods (Bokanowski et al. (2013)), or Gaussian process dynamic programming (Deisenroth et al. (2008)). Different algorithms may be better suited for different problems. It is unlikely that there exists an algorithm that is best in every possible situation, and it is not my objective to propose one. My objective is more modest. I show that the same algorithm can solve a wide range of problems, handles severe nonlinearities, is scalable to large state spaces, is meshfree, requires no pen and paper algebra, demands minimum coding skills, runs on personal computers in reasonable time frames, and is easy to scale up to thousands of computer nodes. Since I made available software to substantiate these claims, I will not delve into exhaustive benchmarks against the aforementioned methods, and I check the solutions with Monte Carlo (MC) simulations or analytical solutions when the latter is available.

The second objective of the paper is to show that the machine learning tools and methods can be used to tackle previously intractable problems in finance and economics and, therefore, point to a new direction for research in computational finance and economics. While the focus in this paper is on continuous-time modeling, I believe that machine learning tools have a much wider scope.

The rest of the paper is organized as follows. Section 2 sets forth the machine learning tools and terminology that will be used to describe the solution method. Section 3 leverages the tools of Section 2 to develop a solution method. We prove that the proposed update rule does not scale with the size of the state space, opening up the possibility of solving nontrivial high-dimensional models in finance and economics. Section 4 takes the method to the laboratory and tests it with controlled experiments. Section 5 concludes.

2 Machine Learning

This section covers the basic machine learning concepts and methods necessary to implement the algorithm in section 3. For excellent textbook treatments, see Goodfellow et al. (2016) and Sutton and Barto (1998). The reader who is already familiar with deep neural networks, stochastic gradient descent (SGD), backpropagation, and generalized policy iteration may skip to the next section.

2.1 Supervised Learning

In a broader sense, the goal of *supervised learning* is to learn how to represent functions. For a concrete example, consider a set of observations $\{X_i, Y_i\}$. We are interested in constructing a function V such that $V(X_i) = Y_i$. X_i could be a picture and Y_i an indicator of whether or not a particular person appears in that picture. A greyscale digital picture with one megapixel, for example, has one million dimensions, so 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 $V(X_i; \Theta)$ and use data to recover Θ.

If one expects a simple linear relation between X_i and Y_i, one could use the familiar normal equation $\Theta = (X'X)^{-1} X'Y$ to construct the parametric representation $V(X_i; \Theta) = X_i'\Theta$. However, linear relations can only go so far. To represent more interesting functions, like the one in the example above, one needs functional forms that are able to capture complex and nonlinear interactions between the regressors. A particularly powerful set of function approximators is the class of neural networks.

2.2 Neural Networks

The starting point of constructing a neural network is the linear model: given a dependent variable $Y_i \in \mathbb{R}$, a data point $X_i \in \mathbb{R}^d$, a vector of coefficients $W_0 \in \mathbb{R}^d$, and a coefficient $b_0 \in \mathbb{R}$, I form the linear combination $Y_i = \langle W_0, X_i \rangle + b_0$, where the $\langle \cdot, \cdot \rangle$ operator denotes the inner product in $\mathbb{R}^d$. A linear model would stop there and look for the parameters W_0 and b_0 that minimize a given loss function. Neural networks go further and apply a *nonlinear* function $\sigma(\cdot)$ on that output. This nonlinear function is known as an *activation function*. Figure 1a shows the rectified linear unit (Jarrett et al. (2009)); this is the default choice in most applications.

Let $H_0 = \sigma(\langle W_0, X_i \rangle + b_0)$ be the output of this nonlinear function, also known as the *hidden unit*. We can perform this operation for a set of vectors and coefficients, $\{W_j, b_j\}_{j \in 0,1,\ldots,n_{H-1}}$, and stack these hidden units into a vector $H \in \mathbb{R}^{n_H}$. Finally, a single-layer neural network takes a linear combination of H to produce the final output: $Y = \langle H, W_{n_H} \rangle + b_{n_H}$. Figure 2a shows a neural network with five hidden units.

It has been shown that, with enough hidden units, one can uniformly approximate any continuous function on compact subsets of $\mathbb{R}^n$, a result known as the *Universal Approximation Theorem* (Cybenko (1989), Hornik (1991)).[7] That result may be familiar to financial economists. Indeed, in the particular case where (i) the activation function $\sigma(\cdot)$ is the rectified linear unit, (ii) the input X is scalar, and (iii) the weights are unit weights ($W_j = 1 \; \forall j$), the output of the j-th hidden unit is the payoff of a call option on X with strike $-b_j$. The output layer, therefore, combines many call options to produce a given payoff. Hence, the result is an extension of the options spanning theorem of Ross (1976), who shows that any contract can be formed as s portfolio of options.

The neural network shown in Figure 2a can be generalized to include many hidden layers, stacked one after the other. In that case, the network is called a *deep neural network*. Figure 2b shows a network with two hidden layers. Empirically, these networks have been found to perform much better than single-layer networks[8].

2.3 Stochastic Gradient Descent

Although the Universal Approximation Theorem guarantees that a rich enough neural network could, in principle, adequately approximate most interesting functions, it does not tell us how to construct such a network. For that step, nearly all applications rely on a variant of the SGD algorithm. The idea is straightforward—a natural measure of fitness for a parametric representation $V(X; \Theta)$ is the one half mean squared error over N observations,

$$MSE\left(\Theta\right) = \frac{1}{2}\mathbb{E}_N\left[\left(V(X;\Theta) - Y\right)^2\right],$$

where $\mathbb{E}_N[\cdot]$ denotes the average over the N observations.

The standard (non-stochastic) method of steepest descent, or simply gradient descent (Cauchy (1847)), moves the parameter Θ in the direction that minimizes the MSE the fastest. In other words, starting with an initial guess Θ, the gradient descent algorithm updates Θ according to

$$\begin{aligned} \Theta &\leftarrow \Theta - \alpha \nabla_\Theta MSE\left(\Theta\right) \\ &= \Theta - \alpha \, \mathbb{E}_N\left[\delta \, \nabla_\Theta V\right], \end{aligned} \tag{1}$$

where α is the *learning rate*, $\nabla_\Theta V$ denotes the gradient operator with respect to the parameter vector Θ, and δ is the *prediction error*, defined as

$$\delta(X_i) \doteq V(X_i; \Theta) - Y_i.$$

The key insight of the stochastic gradient descent is to approximate expectation in equation (1) with a small i.i.d. sample of the data set $\{X_i, Y_i\}$. Thus, for $n \ll N$, we can approximate (1) by[9]

$$\Theta = \Theta - \alpha \, \mathbb{E}_n\left[\delta \nabla_\Theta V\right]. \tag{2}$$

The set of n points used to approximate the gradient is called the *mini-batch*. The use of stochastic methods to compute the MSE loss is one of the aspects that separates machine learning from pure optimization, and it is key to make machine learning operational in high-dimensional problems. One example illustrates this point: consider an estimate of the gradient of the MSE loss based on 100 observations and another based on a 10000 observations. Each iteration based on the latter is 100 times more costly in terms of computational resources, but it only reduces the standard deviation of the gradient by a factor of 10, since the standard error of the mean scales with the square root of the number of observations.[10]

2.4 Backpropagation

The iteration in equation (2) involves all first-order partial derivatives of the network V with respect to its parameters. Therefore, a naïve finite-differences approach to compute the derivatives would be too costly. For example, if the

[7] More precisely, 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.

[8] See Goodfellow et al. (2016), chapter 6 and the references cited therein.

[9] Typical values for n and N are 128 and 1,000,000; for example (Krizhevsky et al. (2012)). For guidelines on how to choose the batch size, see Goodfellow et al. (2016).

[10] This example is from Goodfellow et al. (2016).

network has 100,000 parameters, we would need to compute $V(X_i; \Theta + \varepsilon e_j)$ for every $j \in \{1, \ldots, 100,000\}$, with $\varepsilon \in \mathbb{R}$, and $e_j \in \mathbb{R}^{100,000}$ is the unit vector in the direction of j. Fortunately, machine learning software relies on a more efficient method of computing partial derivatives, called *backpropagation* (Rumelhart et al. (1988)). This algorithm is based on the sequential application of the chain rule, starting from the final layer and moving backward toward 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.[11] Compared to a finite-difference approach applied to the example above, backpropagation provides an economy of five orders of magnitude.

The realization that computing partial derivatives with neural networks is "cheap" was the key development that led to the explosion of interest in neural networks, starting in the 1980s. In the next section, I push this idea further and use backpropagation to compute the exact Bellman residuals.

2.5 Policy Evaluation

Throughout the paper, I assume that infinitely lived agents face a *Markovian Decision Process*; that is, there exists a vector of states $s \in \mathcal{S} \subset \mathbb{R}^n$ that subsumes all relevant information for decision making. An agent following the policy $\pi : \mathcal{S} \to \mathcal{A}$ has expected lifetime utility given by

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} e^{-\rho t} u(\pi(s_t))\right].$$

V_π is called the value function associated with π. Under technical conditions, the value function satisfies the Bellman equation[12]

$$V_\pi(s) = \mathbb{E}\left[u(\pi(s)) + e^{-\rho} V_\pi\left(s'\right)\right], \tag{3}$$

where s' denotes the state vector next period. The right-hand side of equation (3) is the Bellman target and will be denoted by $TV(s)$. *Policy evaluation* is the algorithm for solving the functional equation (3) by turning it into assignments. Starting from an initial arbitrary guess V_π, the value function is updated according to

$$V_\pi(s) \leftarrow TV_\pi(s). \tag{4}$$

The functional equation (3) cannot be exactly solved on a computer, except where the number of states is small. If the number of states is large, as in the case of continuous state variables, all numerical solution methods have to rely on an approximate version of (3). One possibility is to tabulate values for a collection of points in the state space. This is known as the *tabular* or *grid-based* approach. Specifically, we can choose points $\{s_i\}_{i=1}^N$ in the state space and parametrize the value function with $\Theta \in \mathbb{R}^N$ such that $V_\pi(s_i) = \Theta_i$. In that case, the update rule shown in equation (4) is mathematically identical to

$$\Theta \leftarrow \arg\min_{\nu} \frac{1}{2} \mathbb{E}_N\left[(\nu - TV_\pi(\Theta))^2\right]. \tag{5}$$

The minimization on the right-hand side of equation (5) is straightforward in the tabular case. For each state s_i, we set $\Theta_i \leftarrow TV(s_i; \Theta)$ and make the expectation exactly zero. A more general approach, however, is to use the gradient descent to minimize the expectation. An application of the one-step gradient descent produces the following update rule:

$$\begin{aligned} \Theta \leftarrow & \Theta - \alpha\, \mathbb{E}_N\left[(V_\pi - TV_\pi)\nabla_\Theta V\right] \\ = & \Theta - \alpha\, \mathbb{E}_N\left[\delta\, \nabla_\Theta V\right], \end{aligned} \tag{6}$$

where α is the learning rate and $\delta \doteq V_\pi - TV_\pi$. In the appendix E.1, I show that equation (6) is equivalent to (4) when $\alpha = N$. The notation in (6), however, opens up the possibility of using arbitrary function representations, SGD, and backpropagation to perform the Bellman updates. Specifically, we can parametrize the value function with a deep neural network, use backpropagation to compute $\nabla_\Theta V$, and use SGD to estimate the expectation.

[11] See Baydin et al. (2015) for a survey on backpropagation and other automatic differentiation methods.

[12] See Stokey et al. (1989) and Ljungqvist et al. (2000)

2.6 Policy Improvement

Knowing the value function V_π associated with the policy π makes it possible to find a better policy $\pi' : S \to \mathcal{A}$. Indeed, let

$$Q_\pi(s,a) \doteq \mathbb{E}\left[u(a) + e^{-\rho} V_\pi(s')\right],$$

and

$$\pi'(s) \doteq \arg\max_a Q_\pi(s,a). \tag{7}$$

Q_π is the *action-value function*, or *Q-function*, associated with the policy π. The *policy improvement theorem* (Bellman (1957), Howard (1960)) guarantees that $\pi'(s) \geq \pi(s),\ \forall s \in \mathcal{S}$. Equation (7) is therefore called *policy improvement*. The algorithm that alternates policy evaluation (6) and policy improvement (7) is called *policy iteration*. When the action space $\mathcal{A}$ is large, the maximization on the right-hand side of equation (7) 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)).

3 Solution Method

In this section, I show how the machine learning tools presented in Section 2 can be combined with Ito's Lemma to overcome the second curse of dimensionality and solve high-dimensional nonlinear dynamic stochastic problems in continuous time. Since I make explicit use of the environment dynamics, this is not a reinforcement learning algorithm.

To overcome the first curse of dimensionality, I will use deep neural networks to approximate the policy π and its associated value function V_π. In the artificial intelligence (AI) literature, the network that approximates the policy is called *actor* and the network that approximates the value function is called *critic*. We will denote by Θ_A the vector of parameters of the actor and by Θ_C the vector of parameters of the critic.

3.1 A Generalized Policy Iteration Algorithm

With the exception of the main theoretical result, Theorem 1, all derivations are heuristic. We begin with the discrete-time problem presented in Section 2. We write the corresponding equations for an instant Δt into the future, as in Achdou et al. (2014) and Brunnermeier and Sannikov (2016).

Lemma 1. *Let $HJB_\pi(s) \doteq u(\pi(s)) - \rho V_\pi(s) + \frac{\mathbb{E}dV_\pi}{dt}(s)$. For small Δt, the continuous-time Bellman target is approximated by*

$$TV_\pi = V_\pi + \Delta t \cdot HJB_\pi. \tag{8}$$

Plugging equation (8) into the policy evaluation in (6), we arrive at the following value function update rule:

Update rule 1. *Policy evaluation*
Let Θ_C be the vector of parameters of the value function neural network (critic), n the size of a given mini-batch, α_C the learning rate, and $HJB_\pi(s) \doteq u(\pi(s)) - \rho V_\pi(s) + \frac{\mathbb{E}dV_\pi}{dt}(s)$. The critic is updated according to

$$\Theta_C \leftarrow \Theta_C - \alpha_C \, \mathbb{E}_n \left[HJB \nabla_\Theta V_\pi\right]. \tag{9}$$

From equation (8), it is not clear that continuous time offers any advantage over the discrete-time setting. On the one hand, in continuous time, one gets rid of interpolations and integration (the second curse of dimensionality) by trading off expectations for partial derivatives, regardless of the choice of the value function approximator. On the other hand, one has to compute all first- and second-order partial derivatives. This trade-off only makes sense if 1) the state space is small, or 2) there is a efficient way to compute the partial derivatives. Achdou et al. (2014) and Brunnermeier and Sannikov (2016), for instance, present algorithms that perform orders of magnitude faster than their discrete-time counterparts in small-scale problems. It would be hard, however, to scale up these finite-differences algorithms, since each Bellman update involves the computation (and storage) of the Hessian matrix of the value function for every point in the grid. Theorem 1 shows that SGD coupled with backpropagation provides a feasible alternative.

Theorem 1. *Let n_S be the number of Brownian shocks, V a parametric approximation of the value function, and $cost(V)$ the computational cost of evaluating V. The value function drift, $\frac{\mathbb{E}dV}{dt}(s)$, can be calculated with computational complexity of $O(n_S) \cdot cost(V)$.*

Theorem 1 and the update rule 1 provide a way to iterate high-dimensional value functions. Algorithm 1 shows the pseudo-code for policy evaluation in continuous time;

Algorithm 1 Policy evaluation

procedure POLICYEVALUATION(Θ_C) ▷ Update the value function
  Draw $\{s_1, \dots, s_n\}$ random points from the state space
  Compute $\frac{\mathbb{E}dV_\pi}{dt}$ using Theorem 1
  Compute $\nabla_\Theta V_\pi$ with backpropagation
  $\Theta_C \leftarrow \Theta_C - \alpha_C \, \mathbb{E}_n \left[HJB \nabla_\Theta V_\pi \right]$
  return Θ_C ▷ The new neural network representation of V_π

Next, I propose an update rule for the policy network π (the actor). The next lemma is required in the derivation.

Lemma 2. *Let $HJB_\pi(s,a) \doteq u(a) - \rho V_\pi(s) + \frac{\mathbb{E}dV_\pi}{dt}(s,a)$. For small Δt, the continuous-time Q-function $Q(s,a)$ is approximated by*

$$Q_\pi = V_\pi + \Delta t \cdot HJB_\pi. \tag{10}$$

Plugging equation (10) into (7), we obtain the following update rule:

$$\pi'(s) \leftarrow \arg\max_a HJB_\pi(s,a). \tag{11}$$

Finally, applying the stochastic gradient *ascent* on the objective function of equation (11), we derive the update rule for the actor stated next.

Update rule 2. *Policy improvement*
Let Θ_A be the vector of parameters of the policy neural network (actor), n the size of a given mini-batch, α_A the learning rate, and $HJB_\pi(s,a) \doteq u(a) - \rho V_\pi(s) + \frac{\mathbb{E}dV_\pi}{dt}(s,a)$. The actor is updated according to

$$\Theta_A \leftarrow \Theta_A + \alpha_A \, \mathbb{E}_n \left[\nabla_{\Theta_A} HJB(s, \pi(s; \Theta_A)) \right].$$

The pseudo-code for policy improvement in continuous time is shown in Algorithm 2.

Algorithm 2 Policy improvement

procedure POLICYIMPROVEMENT(Θ_A) ▷ Improve the policy function
  Draw $\{s_1, \dots, s_n\}$ random points from the state space
  Compute $\frac{\mathbb{E}dV_\pi}{dt}$ using Lemma 2
  Compute $\nabla_{\Theta_A} HJB(s, \pi(s; \Theta_A))$ with backpropagation
  $\Theta_A \leftarrow \Theta_A + \alpha_A \, \mathbb{E}_n \left[\nabla_{\Theta_A} HJB(s, \pi(s; \Theta_A)) \right]$
  return Θ_A ▷ The new neural network representation of V_π

3.2 Extensions

3.2.1 Recursive preferences

For several decades, time-additive utility functions, such as constant absolute and relative risk aversion utilities, were standard assumptions in most of the economic literature. However, these utilities have unrealistic features that do not properly represent economic agents' behavior in reality. Perhaps their most notable drawback is the fact that time-additive utilities tie the elasticity of intertemporal substitution (a measure of an individual's choice of consumption at different dates) with risk aversion (a measure of individual's choice over risky assets). While the former is inherently a temporal concept once it captures the substitutability of consumption in nearby dates, the latter is an atemporal measure that is well-defined even in single period models.

To overcome this inconsistency, several studies, such as Koopmans (1960), Kreps and Porteus (1978), Epstein and Zin (1989), Weil (1989), Duffie and Epstein (1992), propose a utility formulation that allows agents to have preferece for the timing of uncertainty resolution. The so-called recursive or Epstein-Zin preference gained substantial popularity in the past decades due to its ability to separate risk aversion from elasticity of substitution and to generate realistic empirical regularities observed in financial markets.

The value function of an agent with an Epstein-Zin preference who follows a policy $\pi : \mathcal{S} \to \mathcal{A}$ is given by

$$V_\pi(s) = \left[(1 - e^{-\rho}) \left[(1-\gamma) u(\pi(s)) \right]^{1/\theta} + e^{-\rho} \mathbb{E} \left[(V_\pi(s'))^{1-\gamma} \right]^{1/\theta} \right]^{\theta} \tag{12}$$

where $\theta \doteq (1-\gamma)/(1-1/\psi)$, γ is the relative risk aversion coefficient, and ψ is the elasticity of intertemporal substitution. Defining $J_\pi \doteq (V_\pi)^{1-\gamma}/(1-\gamma)$, we can rewrite equation (12) as

$$J_\pi(s) = \left[(1 - e^{-\rho}) \left[(1-\gamma) u \right]^{1/\theta} + e^{-\rho} \mathbb{E} \left[J_\pi(s') \right]^{1/\theta} \right]^{\theta}. \tag{13}$$

Duffie and Epstein (1992) show that the continuous-time equivalent of the Bellman equation (13) is

$$f + \frac{\mathbb{E}\left[dJ_\pi \right]}{dt} = 0,$$

where

$$f \doteq \rho \theta J \left(\left(\frac{u}{J} \right)^{1/\theta} - 1 \right),$$

is the Duffie-Epstein aggregator. Appendix E.5 shows that it is straightforward to generalize the policy evaluation algorithm to Epstein-Zin preferences.

The update rule with recursive preferences is summarized next.

Update rule 3. *Policy evaluation with EZ preferences*
Let Θ_C be the vector of parameters of the transformed value function $J_\pi(critic)$, n the size of a given mini-batch, α_C the learning rate, and $HJB_\pi(s) \doteq \rho\theta J_\pi \left((u/J_\pi)^{1/\theta} - 1 \right) + \frac{\mathbb{E}[dJ_\pi]}{dt}(s)$. The critic is updated according to

$$\Theta_C \leftarrow \Theta_C - \alpha_C \, \mathbb{E}_n \left[HJB \nabla_\Theta J_\pi \right].$$

3.2.2 Constraints

So far, I allowed both the value function and the policy networks to take any real value. In most applications, however, we would like to impose constraints on the actions that an agent can take. For instance, we may want to impose that labor supply $L \in [0, 1]$ or a short sale constraint $\alpha \geq 0$. In fact, these constraints can be easily incorporated to the method. For instance, we can parametrize the labor network $L : \mathcal{S} \to \mathcal{A}$ as $L(s) = \sigma(DNN(s))$, where DNN is a generic neural network and $\sigma(\cdot)$ is the logistic function. The model in subsection 4.3 provides a concrete example of how constraints can introduce severe nonlinearities to the problem but represent no additional computational challenge for the method.

3.2.3 Using the ergodic set

A key ingredient of reinforcement learning is solving a dynamic problem only in the regions of the state space that *matter*. For instance, in the game of Go, there are more than 10^{100} possible board configurations, yet Google Deepmind's AlphaGo Zero (Silver et al. (2017)), became the world's best player after playing a few million games. This is possible because the vast majority of possible board configurations never realize in real games and are thus irrelevant. Focusing on regions of the state space that are actually visited during simulations (the ergodic set) makes it possible to scale reinforcement learning to large state spaces. The idea is similar to what has been proposed in Judd et al. (2011).

Algorithm 2 is naturally suited for the use of the ergodic set. Instead of sampling the states $\{s_1, \ldots, s_n\}$ according to some pre-specified distribution, we can visit states from previous iterations. In subsection 4.4, I present a high-dimensional problem that exploits the idea of visiting the ergodic set to obtain an accurate numerical solution in a reasonable time frame.

3.2.4 Asset pricing

Stock prices are value functions under the risk-neutral probability. Indeed, no-arbitrage implies the existence of a measure $\mathbb{Q}$ such that

$$P_t = \frac{1}{1+r_f} \mathbb{E}^{\mathbb{Q}} \left[P_{t+1} + D_{t+1} \right] = u + e^{-\rho} \mathbb{E}^{\mathbb{Q}} [P_{t+1}],$$

where P is the (ex-dividend) stock price, D is dividend flow, r_f is the risk-free rate, $u \doteq \frac{1}{1+r_f} \mathbb{E}^{\mathbb{Q}} \left[D_{t+1} \right]$, and $\rho \doteq \log(1 + r_f)$.

In continuous time, it is computationally straightforward to change from the physical to the risk-neutral probability, once we have an expression for the state price density $\xi_t = (\xi_t)_{t\geq 0}$. To fix notation, let B and $B^{\mathbb{Q}}$ be the vectors of Brownian shocks under the physical and risk-neutral probabilities, respectively. Girsanov's Theorem implies that the relationship between B and $B^{\mathbb{Q}}$ is given by

$$dB_t^{\mathbb{Q}} = dB_t - \frac{stoch(d\xi_t)}{\xi_t}dt,$$

where $stoch(d\xi_t)/\xi_t$ denotes the stochastic part of $d\xi_t/\xi_t$ (i.e., the negative of the market prices of risk). We can use Ito's Lemma and backpropagation to compute the volatility of the stochastic discount factor, $stoch(d\xi_t)/\xi_t$. Line 36 of the computer code in Appendix C shows that this change of measure implementation is straightforward, as well as the evaluation of the Bellman iteration.

4 Results

This section presents the results for four experiments of increasing complexity. The first two experiments are canonical asset pricing models and involve only policy evaluations, since the processes for the dividends are exogenously given. The third example is a dynamic stochastic general equilibrium (DSGE) model that has been used to benchmark solution methods in economics. The last experiment is a blueprint for designing new experiments that can be made as complex as the researcher wishes.

All numerical experiments were conducted on a personal computer with the following specifications: MacBook Pro laptop with Mac OS 10.12.6 Sierra, 2.6 GHz Intel Core i5, 8 GB 1600 MHz DDR3, and 256 GB PCle-based storage.

Experiments 1, 2 and 3 ran for 1 minute, while experiment 4 ran for 8 hours on the same computer accelerated by an NVIDIA GTX 1080 Ti GPU.

4.1 Two Trees

We start by illustrating the efficacy and accuracy of the continuous-time policy evaluation algorithm of Section 3 by solving the "two-trees" model of Cochrane et al. (2008).

Despite the simplicity of the economy setup, the model of Cochrane et al. (2008) produces a number of interesting asset pricing results, such as time-varying risk premium, forecastable returns, and "excess volatility." The challenges of solving the problem with standard tools justify the three decades separating the one-tree model of Lucas (1978) and two-trees model of Cochrane et al. (2008). For the particular case of log utility, the authors derive closed-form expressions for the equilibrium quantities, which we use to check the accuracy of the numerical solutions produced by the proposed method. The exposition is kept to a minimum, and I refer the reader to Cochrane et al. (2008) for additional details.

The representative consumer maximizes the following 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 produced by two trees, indexed by $i \in \{1, 2\}$. The exogenous dividend process $D_i = (D_{it})_{t\geq 0}$ satisfies the following stochastic differential equation:

$$\frac{dD_{it}}{D_{it}} = \mu_i dt + \sigma_i dB_{it}.$$

The Brownian shocks B_1 and B_2 have correlation ϱ.

The authors show that the equilibrium quantities, such as the dividend yield, the short-term interest rate, the expected return, and volatility, are driven by a single state variable, namely, the dividend share of the first tree, denoted by s. Figure 3 compares the numerical solution produced by continuous-time value iteration with neural networks approximation and the analytical solution of Cochrane et al. (2008). The graphs show the above-named equilibrium variables as a function of the dividend share. As indicated by the plots, the high nonlinearities displayed by these functions suggest that log linearization methods would fail to capture these curvatures, producing inaccurate numerical solutions. Value iteration, on the other hand, has no difficulty in capturing nonlinear dynamics due to its global nature and the flexibility of neural networks.

Although the model of Cochrane et al. (2008) only admits closed-form solution in the particular case of log utility and dividend dynamics with constant drifts and volatilities, the proposed method works in a much more general setting. In

particular, more general preferences (e.g., the Duffie-Epstein-Zin preference of Epstein and Zin (1989) and Duffie and Epstein (1992) and the habit formation of Campbell and Cochrane (1999)) or time-varying drifts and volatilities (e.g., Bansal and Yaron (2004) and Caldara et al. (2012)) represent no additional challenge to the algorithm, as I illustrate in Section 4.3.

We test the accuracy of the numerical solutions presented in Figure 3 by calculating the associated absolute errors. Figure 4 shows the distribution of the HJB residuals (left panel) and the actual dividend yield errors (right panel). We use a test set consisting of 10,000 random observations drawn uniformly from the set $[10^{-4}, 10^2]$. As is illustrated by the plots, the difference between the dividend yield numerical and analytical solutions is tiny.

4.2 Ten Trees

Martin (2013) extends the setting of Cochrane et al. (2008) by assuming the existence of N trees, the so-called orchard, and allowing for arbitrary relative risk aversion. The author provides integral formulas for the equilibrium quantities that are functions of the $N-1$ dividend shares in the economy. Due to the enlarged state space, the integral formulas are, however, subject to a severe curse of dimensionality, limiting the applicability of the analytical results to setups with three or four trees. We can, nevertheless, test my numerical solution for dividend yields in a much larger orchard economy with the numerical solution obtained by MC simulations.

In this subsection, I test the value iteration algorithm of Section 3 in a Lucas orchard economy with ten trees. The experiment is as follows. First, I construct a test set by drawing random observations of the ten-dimensional state space. For each of the sampled points, I simulate the economy 15,000 times for a period of 150 years and compute the average present value of future dividends. With the parameters presented in Table 2, the computation of the dividend yield for a single point of the state space takes a few days to complete. Using 100 computer nodes, I compute the dividend yield for a test set of 100 points.

Figure 5 compares the MC result with the dynamic programming counterpart. The MC solutions are represented by the 45-degree line in red, with the one standard deviation error band. The value iteration results are reported as blue dots. Once again, dynamic programming produces highly accurate solutions, with a mean error of 0.33% for the price-dividend ratio after five hours of training on a personal computer. Noticeably, virtually all of the value iteration results fall within a one standard deviation error of the MC results, so it is likely that the 0.33% figure is an upper bound on the numerical error.

4.3 A DSGE Model with Stochastic Volatility, EZ Preferences, and Irreversible Investment

In this section, I investigate the performance of the proposed method in a production economy. The model setup is similar to Aruoba et al. (2006) and Caldara et al. (2012), which provide detailed comparisons of different solution techniques to solve DSGE models commonly used in economics. This model is chosen for two reasons. First, the setup is straightforward and familiar to most economists. Second, irreversibility of investment implies that the first-order conditions are valid only as inequalities. As a result, perturbation methods (see Judd and Guu (1997)) produce nontrivial errors for a sufficiently high capital stock.

The model setup is as follows. The central planner chooses aggregate consumption C and labor L to maximize the following expected lifetime utility:

$$\max_{C,L} \mathbb{E}\left[\int_0^\infty e^{-\rho t} \frac{(C_t^\nu L_t^{1-\nu})^{1-\gamma}}{1-\gamma} dt\right],$$

where $\gamma > 0$ is the relative risk aversion, $\rho > 0$ is the time discount preference parameter, and $\nu \in (0,1)$ is the consumption elasticity.

The final good is produced with a Cobb-Douglas technology $Y_t = e^{z_t} K_t^\zeta L_t^{1-\zeta}$, where $\zeta \in (0,1)$ is the capital elasticity, K_t is the capital stock, and z_t is the stochastic productivity, with dynamics

$$dz_t = -(1-\lambda) z_t dt + e^{\sigma_t} dB_{1t},$$

where $-(1-\lambda)$ is the constant expected growth rate of productivity and the process σ_t satisfies the following stochastic differential equation:

$$d\sigma_t = (1-\varphi)(\bar{\sigma} - \sigma_t) dt + \eta dB_{2t}.$$

The Brownian shocks B_1 and B_2 are independent. The positive constants $1-\varphi$, $\bar{\sigma}$, and η are the speed of mean reversion, the long-term mean, and volatility, respectively.

Capital is subject to a constant depreciation rate δ and accumulates according to

$$dK_t = (I_t - \delta K_t)dt,$$

where the investment policy I_t satisfies the resource constraint $I_t = Y_t - C_t$.

The central planner maximization problem is summarized as

$$\begin{aligned} V(K, z, \sigma) = \max_{C,L} \mathbb{E}\left[\int_0^\infty e^{-\rho t} \frac{(C_t^\nu L_t^{1-\nu})^{1-\gamma}}{1-\gamma} dt\right], \\ s.t. \\ dK_t = (e^{z_t} K_t^\zeta L_t^{1-\zeta} - C_t - \delta K_t)dt, \\ dz_t = -(1-\lambda)z_t dt + e^{\sigma_t} dB_{1t}, \\ d\sigma_t = (1-\varphi)(\bar{\sigma} - \sigma_t)dt + \eta dB_{2t}. \end{aligned}$$

The optimal policies C and L are defined implicitly by the equations

$$C, L = \arg\max_{c,l} HJB(K, z, \sigma, c, l),$$

$$HJB(K, z, \sigma, c, l) \equiv u(c, l) - \rho V + \frac{\mathbb{E}dV}{dt}(K, z, \sigma, c, l).$$

Since the labor policy L is restricted to the interval $[0, 1]$, the labor network is parametrized by $L(\cdot) = \sigma(DNN(\cdot))$, where σ is the logistic function and DNN is a generic neural network. Likewise, since investment is irreversible, the savings network is parametrized by $s(\cdot) = \sigma(DNN(\cdot))$. The architecture of the value function and policy networks is shown in Table 3.

Figure 6 compares the numerical solution of the policy iteration algorithm of Section 3 (solid blue line) with the solution of a second- and third-order perturbation methods (dashed red and green lines, respectively). The productivity z and the stochastic volatility σ are held constant at their steady states, and the steady state of capital is indicated by the vertical dashed gray line. A couple of observations are in order. It is well understood that perturbation methods produce good approximations near that steady state of NGMs (see Caldara et al. (2012)). As can be observed in the graph, the policy iteration result coincides with the results from standard perturbation methods. Away from the steady state, however, perturbation methods may produce very inaccurate results.

Figure 6 shows the optimal policies and highlights the main drawback of local methods. Near the steady state, the first-order conditions hold with equality, and perturbation produces accurate results. Away from the steady state and for sufficiently high levels of capital, the central planner hits a constraint. Ideally, in states of the world where capital is abundant, the planner would like to disinvest and increase consumption. But if investment is irreversible, the Euler equation for consumption holds with strict inequality. Therefore, the agent's behavior in that region is completely different from her actions near the steady state. The result is a remarkable failure of local methods in representing such a distinct consumption policy behavior.

Unlike local methods, global methods like the policy iteration of Section 3 do not suffer from this severe shortcoming and can efficiently capture distinct behaviors in different regions of the state space. The bottom plot of Figure 6 provides additional evidence of the accuracy of the value iteration solution. For a sufficiently high level of capital, we can compute analytically the optimal level of labor supply L. Indeed, for high values of K, the representative agent consumes the entire output Y. Therefore, the optimal labor supply satisfies $L = \arg\max L^{(1-\zeta)\nu}(1-L)^{1-\nu} \approx 0.283$, which is the value found by the policy network.

4.4 NGM as a Laboratory

Testing algorithms on benchmark problems that admit closed-form solutions is crucial to learn about their strengths and limitations, to assess their accuracies, and to suggest improvements. Unfortunately, most finance and economics problems that admit analytical solutions are simple and therefore do not pose a significant challenge to advanced methods.

Ideal tests need to satisfy two criteria. First, the problem at hand has to be complex enough to preclude the use of standard solution techniques, like log linearizations or projection methods. Second, the problem has to admit closed-form solution, so we can check the accuracy.

In this subsection, I propose a way to generate problems that satisfy these two conditions by construction. This provides a testbed to evaluate existing solution methods and serves as a laboratory to develop new ones.

Consider an NGM, where the representative agent chooses consumption and investment to maximize her expected lifetime utility. There are $N+1$ state variables: N exogenous variables $(x_1, \ldots, x_N)$ affecting the aggregate productivity, and the capital stock K. To ease notation, I suppress the time index and summarize the model as

$$\begin{aligned} V(K, x_1, \ldots, x_N) = \max_C \mathbb{E}\left[\int_0^\infty e^{-\rho t} u(C) dt\right], \\ s.t. \\ dK = (Y - C)dt, \\ Y = A(x_1, \ldots, x_N)K, \\ dx_i = \mu_i dt + \sigma_i dB_i, \quad \forall i \in \{1, \ldots, N\}. \end{aligned}$$

The associated Bellman equation becomes

$$\begin{aligned} 0 = u(C^*) - \rho V + \frac{\mathbb{E}dV}{dt}, \\ C^* = V_K^{-1/\gamma}. \end{aligned}$$

In addition, it can be shown that the value function satisfies

$$V(K, x_1, x_2, \ldots, x_N) = \frac{K^{1-\gamma}}{1-\gamma}\phi(x_1, \ldots, x_N).$$

Thus, the Bellman equation can be rewritten as

$$A(x_1, ..., x_N) = \frac{1}{K}\left(\frac{1}{V_K^{1/\gamma}} - \frac{1}{V_K}\left(\frac{V_K^{1-1/\gamma}}{1-\gamma} - \rho V + \frac{K^{1-\gamma}}{1-\gamma}\frac{\mathbb{E}d\phi}{dt}\right)\right). \tag{14}$$

Equation (14) implies that, for a given function ϕ, one can reverse engineer the productivity $A(\cdot)$ such that the Bellman equation holds exactly. Therefore, we can postulate an arbitrary value function and test if a given algorithm is able to produce an accurate solution by analyzing its error relative to the analytical solution. Naturally, this setup can and should be used to develop and to improve methods and practices.

To illustrate the methodology, consider the function $\phi : \mathbb{R}^{10} \to \mathbb{R}$, defined as

$$\phi(\mathbf{x}) = \frac{1}{10}\sum_{j=1}^{10} e^{-\|\mathbf{x} - \mathbf{L}_j\|^2}, \tag{15}$$

where the landmarks $\mathbf{L}_j \in \mathbb{R}^{10}$ are drawn uniformly from the state space. I emphasize that any function can be used to test solution methods and that my choice of the functional form presented in (15) is due to the simplicity of coding it and its high curvatures in every dimension, as illustrated by Figure 7. To highlight the nonlinearities induced by this functional form, the plot shows five different cross sections of the true ten-dimensional value function.

The left plot of Figure 8 compares the analytical value function (red line) with the neural network solution obtained by iterating the Bellman equation (blue line). The relative errors of the value function and the implied optimal consumption are presented in the right plot of Figure 8. The graph shows the distribution of the errors evaluated on a test set of one million points, sampled uniformly from the set $[-2, 2]^{10}$. The mean consumption error is less than 0.1%, and the maximum error is 1.2%.

Figure 9 plots the learning curves for this experiment that is, the time series of the errors during training. The plot on the left shows the mean and maximum errors for the value function and consumption at different time stamps, evaluated at random batches of 5000 points. The plot on the right shows the evolution of the HJB residuals and the mean squared errors of the Bellman iteration.[13]

[13] As in every machine learning algorithm, it is important to monitor the learning curves during training to decide when to stop. After enough iterations, the learning curve typically flattens out. At this point, one may try reducing the learning rate or increasing the mini match size in order to refine the solution. If additional tuning of the hyperparameters doesn't improve learning, the training process is stopped.

5 Conclusion

This paper proposes a novel numerical method that alleviates the three curses of dimensionality (Powell (2007)). We show that the method can solve complex continuous-time models in macroeconomics and finance, eliminating the long-lasting bottleneck in computational finance that prevented researchers from studying large nonlinear dynamical systems for decades. The method rests on two pillars: (i) using deep learning to overcome the first and third curses of dimensionality; (ii) combining Ito's Lemma with backpropagation overcome the second curse of dimensionality.

We illustrate the applicability of the method by solving nontrivial benchmark models in financial economics, and I show that the method produces accurate results in reasonable running times on a personal computer.

The ability to globally solve nonlinear models in high-dimensional state spaces is crucial to study other first-order problems in macroeconomics and finance that are, as of now, intractable. Examples of such topics include, but are not limited to, the dynamics of risk premia, endogenous systemic risk, and unconventional monetary policy. This paper proposes a way forward by leveraging cutting-edge machine learning technology.

Programming a deep neural network and computing its gradient with respect to thousands of parameters may seem like a daunting task, and it may give the impression that one needs advanced knowledge of computer science and applied mathematics to implement the method proposed in this paper. This is not the case. Indeed, these operations constitute the building blocks of most machine learning commercial applications and research. As such, tech giants like Google, Facebook[14] and Microsoft[15] have all invested heavily in the development of open-source software that automatically does much of this heavy lifting. As a result, the barriers to entry into machine learning have become quite low in the past couple of years.

In addition, these professional libraries are optimized for performance and efficiency. Machine learning software is designed to make the best use of the computer's hardware. The same code running on a personal computer can be run on a cluster with thousands of processors with little modification. As they are at the center of the economy of the 21st century, it is reasonable to expect that these technologies will only become more powerful. Bringing economics and finance closer to the machine learning technological frontier is a solid bet not only in current technologies but also in future developments in software, hardware, and methods.

References

Abu-Khalaf, M., Lewis, F. L., 2005. Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network hjb approach. Automatica 41, 779–791.

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

Aruoba, S. B., Fernandez-Villaverde, J., Rubio-Ramirez, J. F., 2006. Comparing solution methods for dynamic equilibrium economies. Journal of Economic Dynamics and Control 30, 2477–2508.

Bansal, R., Yaron, A., 2004. Risks for the long run: A potential resolution of asset pricing puzzles. The Journal of Finance 59, 1481–1509.

Barro, R. J., 2006. Rare disasters and asset markets in the twentieth century. The Quarterly Journal of Economics 121, 823–866.

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

Bellman, R., 1957. Dynamic Programming. Princeton University Press, Princeton, NJ, USA, first ed.

Bertsekas, D. P., Tsitsiklis, J. N., 1996. Neuro-Dynamic Programming. Athena Scientific, first ed.

Bokanowski, O., Garcke, J., Griebel, M., Klompmaker, I., 2013. An adaptive sparse grid semi-lagrangian scheme for first order hamilton-jacobi bellman equations. Journal of Scientific Computing 55, 575–605.

Brunnermeier, M., Sannikov, Y., 2016. Chapter 18 - macro, money, and finance: A continuous-time approach. Elsevier, vol. 2 of *Handbook of Macroeconomics*, pp. 1497 – 1545.

[14] http://pytorch.org/

[15] https://docs.microsoft.com/en-us/cognitive-toolkit/

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

Byrd, R. H., Lu, P., Nocedal, J., Zhu, C., 1995. A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Comput. 16, 1190–1208.

Caldara, D., Fernández-Villaverde, J., Rubio-Ramírez, J. F., Yao, W., 2012. Computing {DSGE} models with recursive preferences and stochastic volatility. Review of Economic Dynamics 15, 188 – 206.

Campbell, J. Y., Cochrane, J. H., 1999. By force of habit: A consumption-based explanation of aggregate stock market behavior. Journal of Political Economy 107, 205–251.

Campbell, J. Y., Shiller, R. J., 1988. The dividend-price ratio and expectations of future dividends and discount factors. The Review of Financial Studies 1, 195–228.

Campbell, J. Y., Viceira, L. M., 1999. Consumption and portfolio decisions when expected returns are time varying*. The Quarterly Journal of Economics 114, 433–495.

Cauchy, A., 1847. Méthode générale pour la résolution des systemes d'équations simultanées. Comp. Rend. Sci. Paris 25, 536–538.

Cochrane, J. H., Longstaff, F. A., Santa-Clara, P., 2008. Two trees. Review of Financial Studies 21, 347–385.

Coulom, R., 2004. High-accuracy value-function approximation with neural networks applied to the acrobot. In: *ESANN 2004, 12th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 28-30, 2004, Proceedings*, pp. 7–12.

Cybenko, G., 1989. Approximation by superposition of sigmoidal functions. Mathematics of Control, Signals and Systems 2, 303–314.

Deisenroth, M. P., Peters, J., Rasmussen, C. E., 2008. Approximate dynamic programming with gaussian processes. In: *2008 American Control Conference*, pp. 4480–4485.

Duffie, D., 2010. Dynamic asset pricing theory. Princeton University Press.

Duffie, D., Epstein, L. G., 1992. Stochastic differential utility. Econometrica: Journal of the Econometric Society pp. 353–394.

Epstein, L. G., Zin, S. E., 1989. Substitution, risk aversion, and the temporal behavior of consumption and asset returns: A theoretical framework. Econometrica 57, 937–969.

Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y. W., Titterington, M. (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, PMLR, Chia Laguna Resort, Sardinia, Italy, vol. 9 of *Proceedings of Machine Learning Research*, pp. 249–256.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep learning. MIT press.

Haugh, M. B., Kogan, L., 2004. Pricing american options: A duality approach. Operations Research 52, 258–270.

He, Z., Krishnamurthy, A., 2013. Intermediary asset pricing. The American Economic Review 103, 732–770.

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

Hennessy, C. A., Whited, T. M., 2005. Debt dynamics. The Journal of Finance 60, 1129–1165.

Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural Networks 4, 251 – 257.

Howard, R. A., 1960. Dynamic Programming and Markov Processes. MIT Press, Cambridge, MA.

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR abs/1502.03167.

Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al., 2009. What is the best multi-stage architecture for object recognition? In: *Computer Vision, 2009 IEEE 12th International Conference on*, IEEE, pp. 2146–2153.

Judd, K. L., 1992. Projection methods for solving aggregate growth models. Journal of Economic Theory 58, 410 – 452.

Judd, K. L., Guu, S.-M., 1997. Asymptotic methods for aggregate growth models. Journal of Economic Dynamics and Control 21, 1025–1042.

Judd, K. L., Maliar, L., Maliar, S., 2011. Numerically stable and accurate stochastic simulation approaches for solving dynamic economic models. Quantitative Economics 2, 173–210.

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

Kingma, D., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Kogan, L., Papanikolaou, D., 2013. Firm characteristics and stock returns: The role of investment-specific shocks. The Review of Financial Studies 26, 2718–2759.

Koopmans, T. C., 1960. Stationary ordinal utility and impatience. Econometrica: Journal of the Econometric Society pp. 287–309.

Kreps, D. M., Porteus, E. L., 1978. Temporal resolution of uncertainty and dynamic choice theory. Econometrica 46, 185–200.

Krizhevsky, A., Sutskever, I., Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C. J. C., Bottou, L., Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., pp. 1097–1105.

Krueger, D., Kubler, F., 2004. Computing equilibrium in olg models with stochastic production. Journal of Economic Dynamics and Control 28, 1411 – 1436.

Lei Ba, J., Kiros, J. R., Hinton, G. E., 2016. Layer Normalization. ArXiv e-prints .

Levenberg, K., 1944. A method for the solution of certain non-linear problems in least squares. Quarterly of Applied Mathematics 2, 164–168.

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

Ljungqvist, L., Sargent, T., Institution, H., 2000. Recursive Macroeconomic Theory. MIT Press.

Lucas, R. E., 1978. Asset prices in an exchange economy. Econometrica 46, 1429–1445.

Marquardt, D. W., 1963. An algorithm for least-squares estimation of nonlinear parameters. Journal of the Society for Industrial and Applied Mathematics 11, 431–441.

Martin, I., 2013. The lucas orchard. Econometrica 81, 55–111.

McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., Chikkerur, S., Liu, D., Wattenberg, M., Hrafnkelsson, A. M., Boulos, T., Kubica, J., 2013. Ad click prediction: A view from the trenches. In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, New York, NY, USA, KDD '13, pp. 1222–1230.

Mhaskar, H., Liao, Q., Poggio, T. A., 2016. Learning real and boolean functions: When is deep better than shallow. CoRR abs/1603.00988.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.

Montufar, G. F., Pascanu, R., Cho, K., Bengio, Y., 2014. On the number of linear regions of deep neural networks. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27*, Curran Associates, Inc., pp. 2924–2932.

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

Pohl, W., Schmedders, K., Wilms, O., 2018. Higher order effects in asset pricing models with long-run risks. The Journal of Finance 73, 1061–1111.

Powell, W. B., 2007. Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics). Wiley-Interscience, New York, NY, USA.

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

Rumelhart, D. E., Hinton, G. E., Williams, R. J., 1988. Neurocomputing: Foundations of research. MIT Press, Cambridge, MA, USA, chap. Learning Representations by Back-propagating Errors, pp. 696–699.

Sannikov, Y., 2008. A continuous- time version of the principal: Agent problem. The Review of Economic Studies 75, 957–984.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D., 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489.

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

Stokey, N., Lucas, R., Prescott, E., 1989. Recursive Methods in Economic Dynamics. Harvard University Press.

Sutton, R. S., Barto, A. G., 1998. Introduction to Reinforcement Learning. MIT Press, Cambridge, MA, USA, first ed.

Tassa, Y., Erez, T., 2007. Least squares solutions of the hjb equation with neural network value-function approximators. IEEE Transactions on Neural Networks 18, 1031–1041.

Vamvoudakis, K. G., Lewis, F. L., 2010. Online actor–critic algorithm to solve the continuous-time infinite horizon optimal control problem. Automatica 46, 878 – 888.

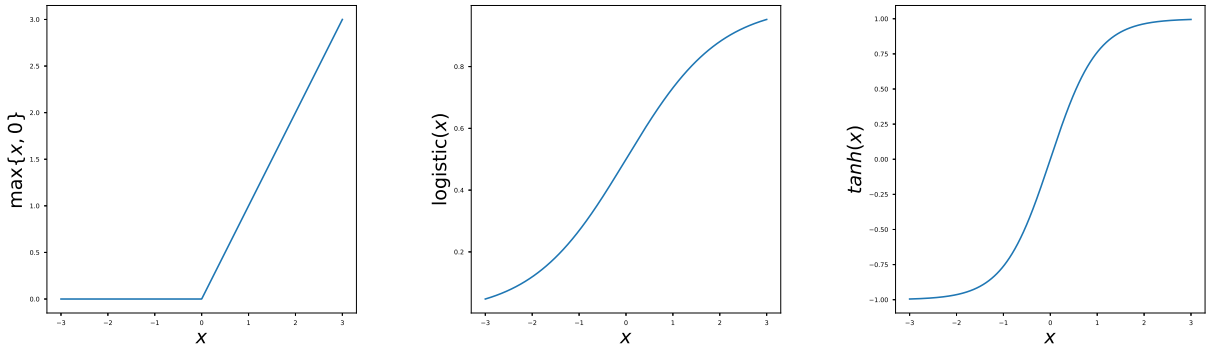
Weil, P., 1989. The equity premium puzzle and the risk-free rate puzzle. Journal of Monetary Economics 24, 401–421.

Winschel, V., Kratzig, M., 2010. Solving, estimating, and selecting nonlinear dynamic models without the curse of dimensionality. Econometrica 78, 803–821.

Zeiler, M. D., 2012. ADADELTA: an adaptive learning rate method. CoRR abs/1212.5701.

A Figures

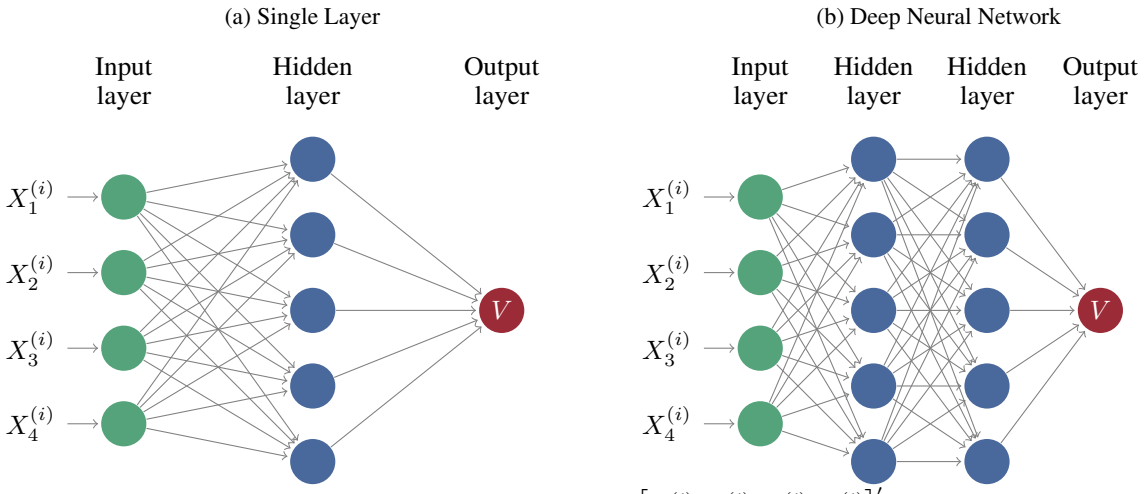
Figure 1: Activation functions



(a) (b) (c)

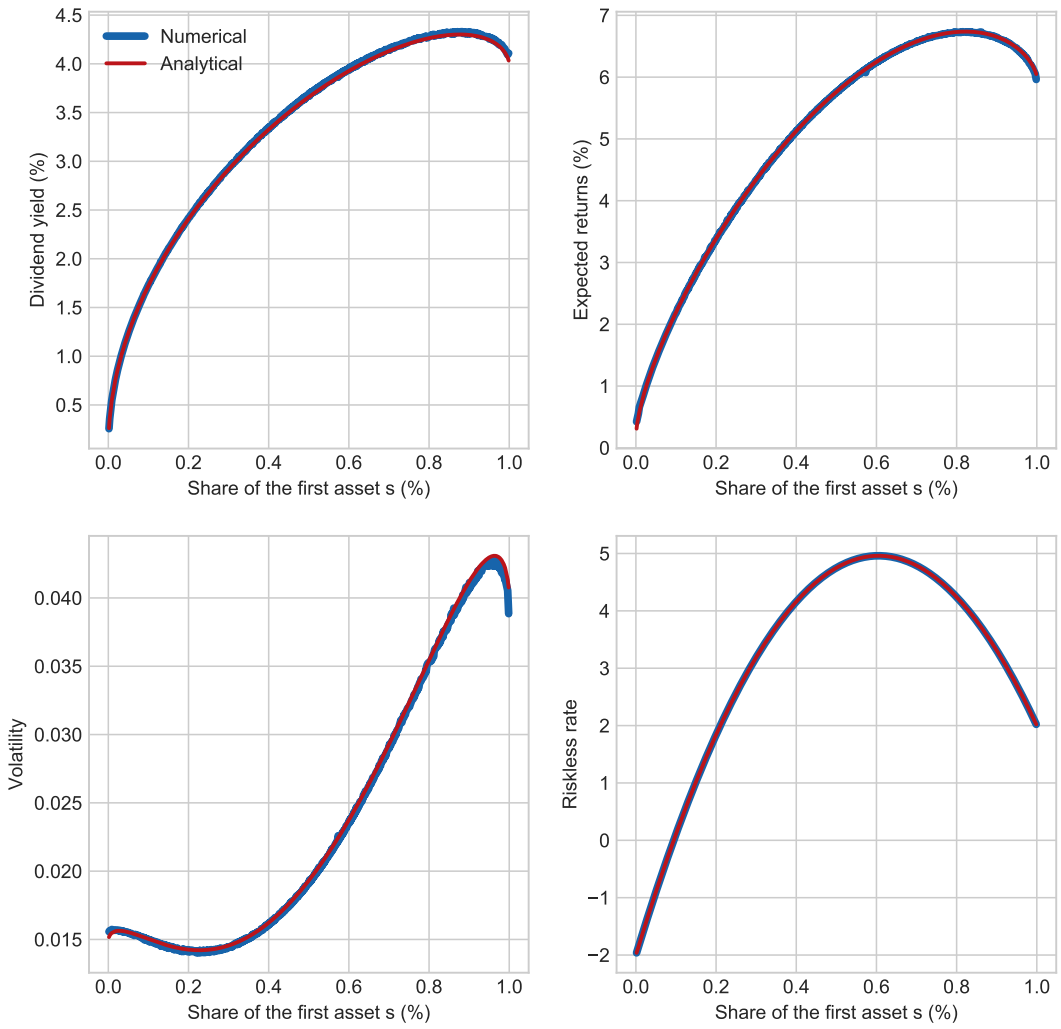
Panel (a) shows the rectified linear unit (ReLu), the most common activation function used in machine learning applications. It is loosely based on how actual neurons behave. For inputs below a certain threshold, the neuron does not fire. For a sufficiently high input, however, the neuron produces a roughly linear signal. The simple form of the ReLu makes it computationally less costly than the alternatives, that typically involve exponentiations. For applications that involve the computation of the second order derivatives, however, the ReLu activation function is not suitable. 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



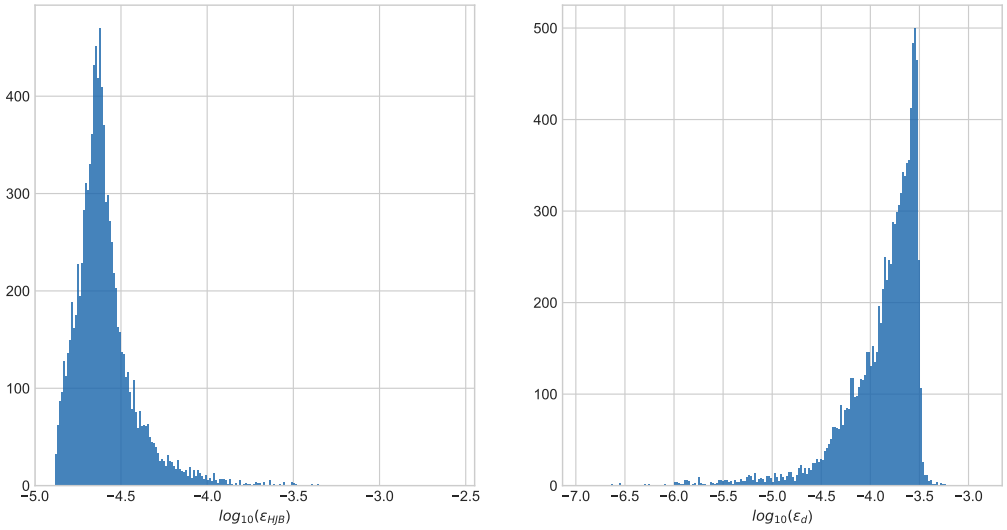
The green circles represent each entry of the input vector $X^{(i)} = \left[X_1^{(i)}, X_2^{(i)}, X_3^{(i)}, X_4^{(i)}\right]'$. The hidden units are represented by purple circles. Each hidden unit perform a composition of a nonlinear function (activation function) and a linear transformation of the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

Figure 3: Two Trees



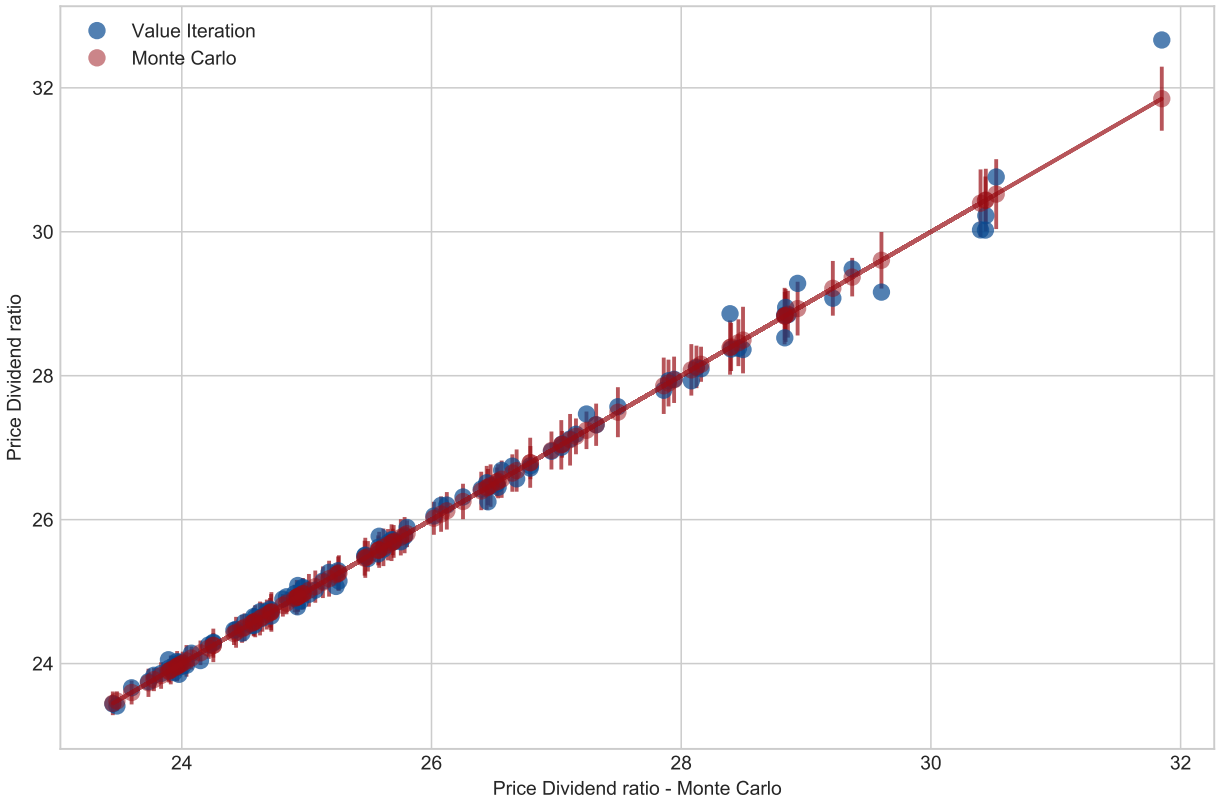
The figure shows the plots of the dividend yield, expected returns, instantaneous volatility, and riskless rate as a function of the first tree dividend share. The blue lines correspond to the numerical solutions and the red lines correspond to the analytical solutions evaluated at the random test set. The parameters values are as follows: $\rho = 0.04$, $\gamma = 1$, $\varrho = -0.5$, $\mu_1 = 0.02$, $\mu_2 = 0.03$, $\sigma_1 = 0.2$ and $\sigma_2 = 0.3$. The neural networks for the normalized asset prices $V = P \cdot (D_1 + D_2)^{-\gamma}$ have two fully-connected hidden layers with 30 hidden units each, and hyperbolic tangent activation functions. Each iteration step is performed on a random batch of 1000 points. The learning rate is set at 10^{-5}.

Figure 4: Two Trees - Distribution of errors



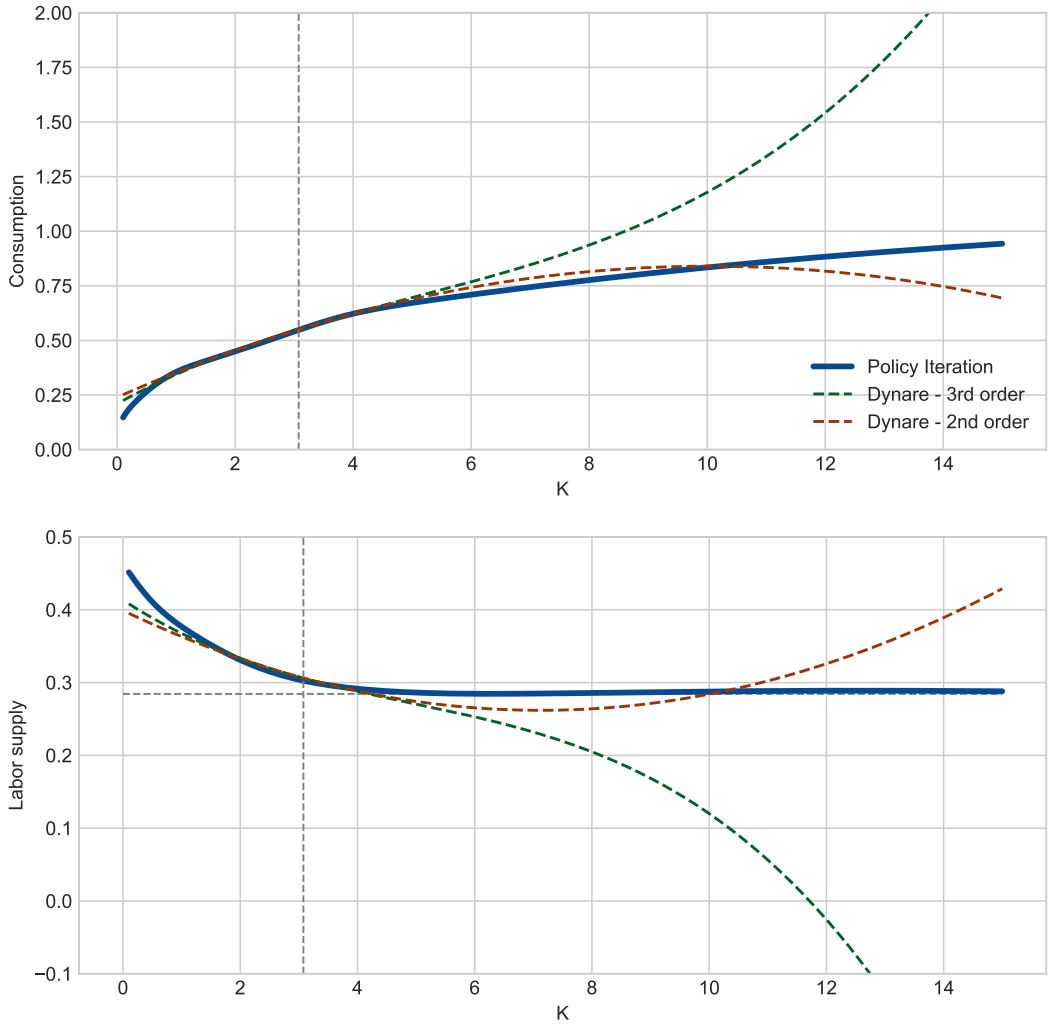
The plot on the left shows the distribution of the ($\log_{10}$) HJB residuals for a test set of 10,000 randomly drawn points $(D_1, D_2) \in [10^{-4}, 100]^2$. The plot on the right shows the distribution of the actual dividend yield errors, i.e., the absolute difference between the numerical and the analytical solution, or simply, $\varepsilon_d = \log_{10} |d^{\text{numerical}} - d^{\text{analytical}}|$.

Figure 5: 10 Trees



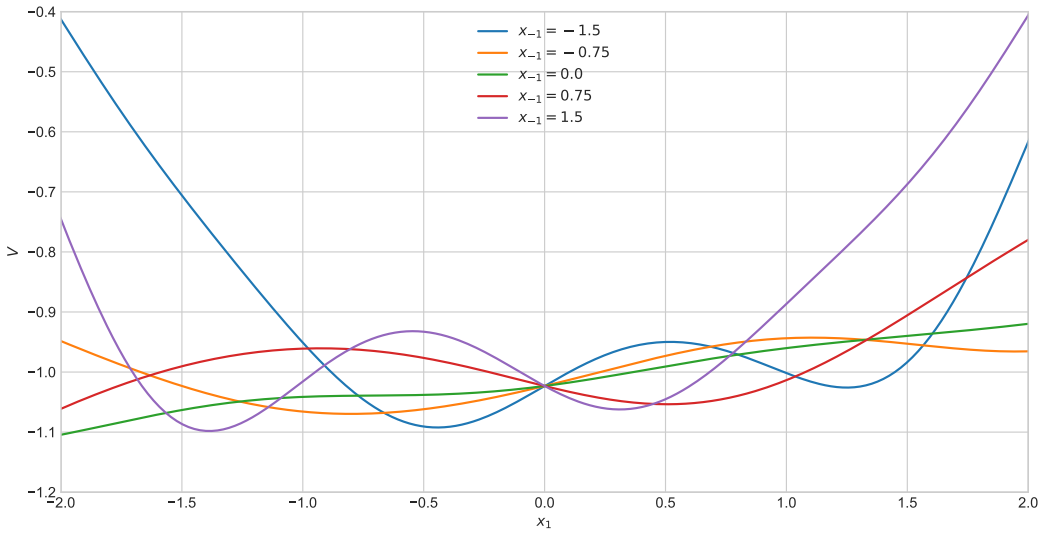
The figure shows a scatter plot of the results produced by dynamic programming (blue) against the Monte Carlo results (red). Monte Carlo results are shown with the one standard deviation error band. The test set is a sample of 100 points drawn uniformly from the set $(D_1, \ldots, D_{10}) \in [0.005, 150]^{10}$. The Monte Carlo results are the average of 15,000 simulated paths spanning 150 years. Time is discretized with $dt = 0.001$. Relative risk aversion, the time discount parameter, dividends' drifts and volatilities are, respectively, $\gamma = 1, \rho = 0.04, \mu_i = 0.2, \sigma_i = 0.02$. All Brownian shocks are independent. The Monte-Carlo results are generated with 100 computer nodes during a period of approximately 60 hours. The dynamic programming results are generated with 8 hours of training on a personal computer.

Figure 6: Cross section of the policy functions



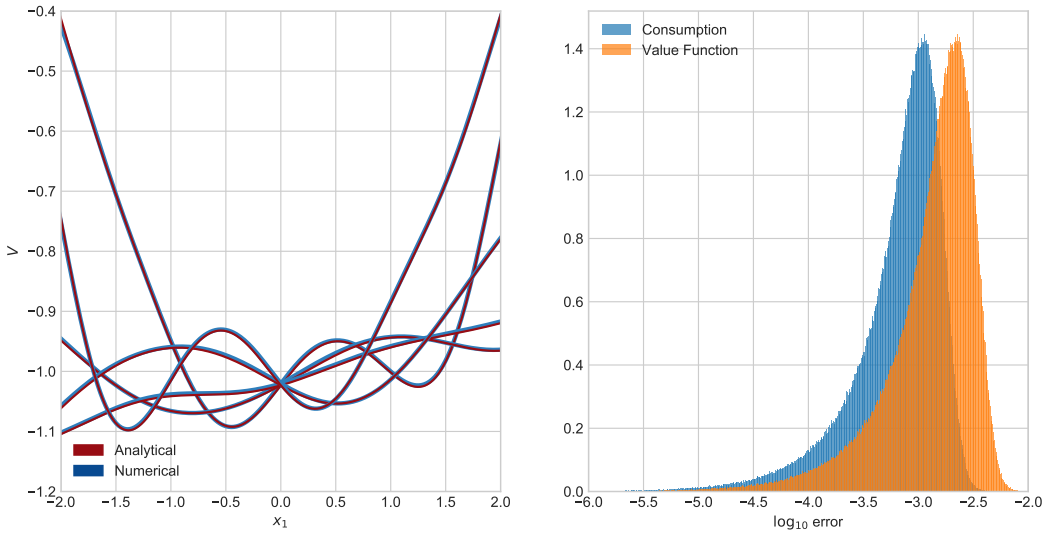
The figure shows the policy functions C and L for $z = 0$, $\sigma = \bar{\sigma}$, $\gamma = 2$, $\rho = 0.04$, $\nu = 0.36, 0.30$, $\lambda = 0.95, -3.86$, $\eta = 0.1$, and $\varphi = 0.9$. The blue line is the output of the policy iteration algorithm with neural networks. The red and green dashed lines are the solutions obtained with perturbation methods, plotted here for comparison. The vertical gray dashed line corresponds to the capital steady state. The horizontal dashed gray line corresponds to the analytical value for L when $K \to \infty$.

Figure 7: Cross section of a ten dimensional value function



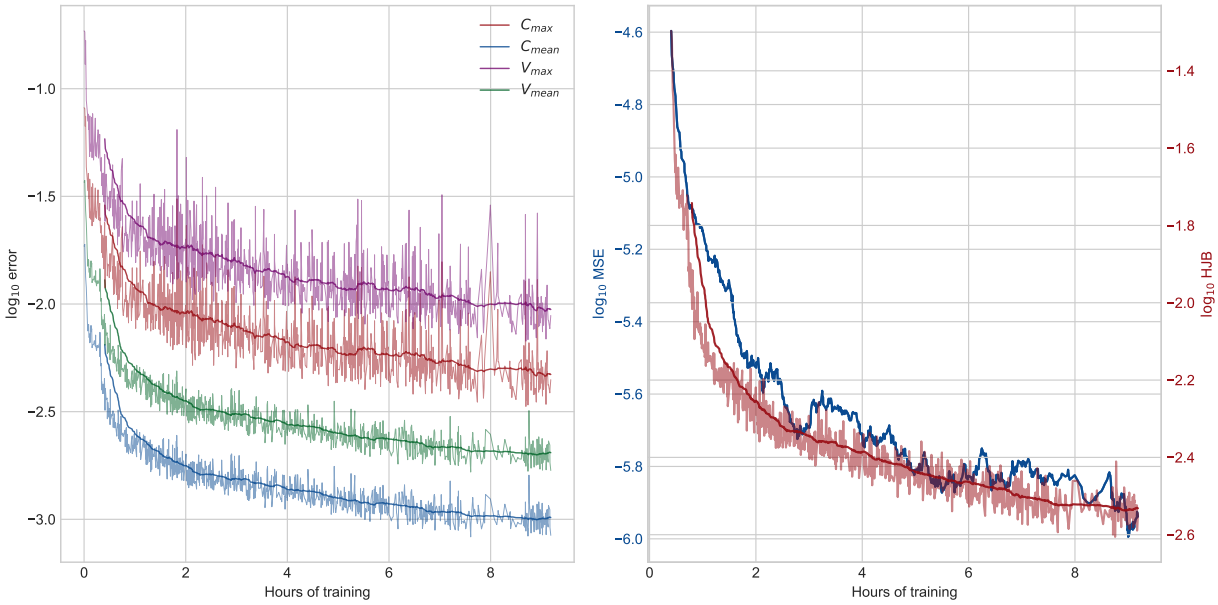
The figure shows a cross section of the value function $\phi(\mathbf{x}) = \frac{1}{10}\sum_{j=1}^{10} e^{-\|\mathbf{x}-\mathbf{L}_j\|^2}$ when the first coordinate varies, and all other coordinates are fixed at $\mathbf{x_{-1}}$. The blue curve, for example, represents $V(x_1)$, for $x_1 \in [-2, 2]$ and $x_2 = x_3 = \ldots = x_{10} = -1.5$.

Figure 8: Numerical vs analytical solution: ten dimensional value function



The plot on the left compares the numerical with the analytical solution for a cross section of the value function $\phi(\mathbf{x}) = \frac{1}{10}\sum_{\mathbf{j=1}}^{\mathbf{10}} \mathbf{e}^{-\|\mathbf{x}-\mathbf{L_j}\|^2}$ when the first coordinate varies, and all other coordinates are fixed at $\mathbf{x_{-1}}$. The blue curve, for example, represents $V(x_1)$, for $x_1 \in [-2, 2]$ and $x_2 = x_3 = \ldots = x_{10} = -1.5$. The plot on the right shows the distributions of the $\log_{10}$ errors for consumption and the value function evaluated at a random test set. The test set consists of 1 million points, drawn uniformly from $[-2, 2]^{10}$.

Figure 9: Learning curves



The figure shows the time series of the errors during training for the ten-dimensional AK model with postulated value function $\phi(\mathbf{x}) = \frac{1}{10}\sum_{j=1}^{10} e^{-\|\mathbf{x}-\mathbf{L}_j\|^2}$. The left panel shows the mean and maximum error for the value function and consumption at different iteration steps, evaluated at random batches of 5000 points. The right panel shows the evolution of the HJB residuals and the mean squared errors of the Bellman iteration.

B Tables

Table 1: Parameters and Network Architecture: Two Trees

	Parameter	Value
	Model parameters	
Time discount	ρ	0.04
Expected dividend growth, first tree	μ_1	0.02
Expected dividend growth, second tree	μ_2	0.03
Dividend volatility, first tree	σ_1	0.20
Dividend volatility, second tree	σ_2	0.30
Dividends correlation	ϱ	-0.50
	Hyperparameters	
Numer of hidden layers		2
Hidden units		[30, 30]
Activation Function		$\tanh$
Type gradient descent		RMSProp
Learning rate		1e-5
Momentum		.99
Mini batch size		1000

Table 2: Parameters and Network Architecture: Ten Trees

	Parameter	Value
	Model parameters	
Coefficient relative risk aversion	γ	1.00
Time discount	ρ	0.04
Expected dividend growth	μ	0.02
Dividend volatility	σ	0.20
	Hyperparameters	
Numer of hidden layers		3
Hidden units		[512, 128, 32]
Activation Function		$\tanh$
Type gradient descent		RMSProp
Learning rate		1e-5
Momentum		.99
Mini batch size		256

Table 3: Hyper parameters and network architecture: NGM Lab

	Parameter	Value
	Model parameters	
Coefficient relative risk aversion	γ	2.00
Time discount	ρ	0.05
Autocorrelation	μ	-0.1
State variable volatility	σ	0.05
Number of layers		3
Hidden units		[128, 32, 16]
Activation function		tanh
Type of gradient descent		RMSProp
Learning rate		1e-5
Mini batch size		256
Momentum		.99

C Full Code for the Two Trees Model (Cochrane et al. (2008))

```
import mlecon as mle
from numpy import sqrt
# ———————— Setup ————————————————————
mle.clear()
# Hyper-parameters
hidden = [64, 32]   # number of units in each hidden layer
Δt = 0.1
dt = mle.dt
t = mle.t

# State space
D1, D2 = mle.states(2)   # Create 2 state variables
dZ = mle.brownian_shocks(2)

# Bounds
bounds = {D1: [1e-4, 100], D2: [1e-4, 100]}   # bounds where D will be drawn
sample = mle.sampler(bounds)   # Op that performs the sampling

# Function approximator
P = mle.network([D1, D2], name='VF', hidden=hidden)

# ———————— Economic Model ————————————————-
γ, ρ, σ1, σ2, μ1, μ2, ϱ = 2, 0.05, 0.2, 0.3, 0.05, 0.07, -0.5   # Parameters

C = D1 + D2                  # Aggregate consumption
M = C**(-γ)                  # Marginal Utility
π = mle.exp(-ρ * t) * M   # SDF

# Dynamics
D1.d = μ1 * D1 * dt + σ1 * D1 * dZ[0]
D2.d = μ2 * D2 * dt + ϱ * σ2 * D2 * dZ[0] + sqrt(1 - ϱ**2) * σ2 * D2 * dZ[1]

# SDF
rf, η = -π.diff / π

HJB = D1 - rf * P + P.drift(new_measure=η)
T = P + Δt * HJB

policy_evaluation = mle.fit(P, T)
env = mle.environment(P)
# ———————— Iteration ————————————————————
program = {sample: 1, policy_evaluation: 1}
env.iterate(program, T='00:02:00')   # Iterate for 2m
```

Listing 1: Two Trees Model

D Full Code for the DSGE Model with Stochastic Volatility, EZ Preferences, and Irreversible Investment (Caldara et al. (2012))

```
import mlecon as mle

mle.clear()    # Reset graph
hidden = [10] * 3   # Network architecture
Δt = 0.1            # dt for simulations
mle.delta_t = Δt

K, z, σ = mle.states(3)         # State variables
dZ = mle.brownian_shocks(2)     # Brownian shocks

# Model parameters
ρ, ν, ζ, δ, λ, σ_, γ, η, φ, ψ = .04, .36, .3, .0196, .95, −3, 4, .1, .9, .5
θ = (1 − γ) / (1 − 1 / ψ)

bounds = {K: [.1, 12.], z: [−.5, .5], σ: [−6, −1]}
sample = mle.sampler(bounds)    # Op. that draws (unif.) random samples from the state space

# Function approximators
inputs = [K, z, σ]
J = mle.network(inputs, hidden, name='VF')
L = mle.network(inputs, hidden, name='labor', bnds=[1e−6, 1])
s = mle.network(inputs, hidden, name='savings', bnds=[1e−6, .999])
# ———— Economic Model ————————————————————————
Y = mle.exp(z) * K**ζ * L**(1 − ζ)
C = (1 − s) * Y
U = (C**ν * (1 − L)**(1 − ν))**(1 − γ) / (1 − γ)
I = s * Y
dt = mle.dt
K.d = (I − δ * K) * dt
z.d = −(1 − λ) * z * dt + mle.exp(σ) * dZ[0]
σ.d =  (1−φ) * (σ_ − σ) * dt + η * dZ[1]

f = ρ * θ * J * ((U / J)**(1 / θ) − 1)              # Duffie-Epstein aggregator
HJB = f + J.drift                                   # HJB residual
T = J + Δt * HJB                                    # Bellman target

policy_eval = mle.fit(J, T)                         # pol. evaluation
policy_improv = mle.greedy(HJB, actions=[s, L])     # pol. improvement
# ———— Iteration ——————————————————————————————
env = mle.environment([J, s, L], gpu=False)                 # Launch graph
program = {sample: 1, policy_eval: 1, policy_improv: 1}
env.iterate(program, T='00:03:00')                          # Iterate for 3 mins
env.save()
```

Listing 2: DSGE model with EZ preferences and stochastic volatility

E Derivations

E.1 Value function iteration is a particular case of Bellman updates with gradient descent

In the tabular case, the value function representation is $V = \sum_{i=1}^{n} \Theta_i e_i$ and the Bellman target is represented by $TV = \sum_{i=1}^{n} t_i e_i$, where $e_i = [0, \ldots, 1, \ldots, 0]$. Therefore $\nabla_\Theta V = \boldsymbol{e} = [1, \ldots, 1]$. . Then,

$$\begin{array}{rcl} V - TV & = & \sum_{i=1}^{n} (\Theta_i - t_i) e_i \\ & \Rightarrow & \\ (V - TV)\nabla_\Theta V & = & \sum_{i=1}^{n} (\Theta_i - t_i) e_i \\ & \Rightarrow & \\ N\mathbb{E}_N[\delta \nabla_\Theta V] & = & \sum_{i=1}^{n} (\Theta_i - t_i) e_i \end{array} \tag{16}$$

It follows that if the learning rate is set at $\alpha = N$, the update rule equation (6) reduces to equation (4).

E.2 Proof of Lemma 1: Bellman target in continuous time

The heuristic derivation of the Bellman operator in continuous time mimics the derivation of the Hamilton Bellman Equation (HJB) that can be found in most asset pricing textbooks.[16] The only difference is the very last step, where we do not cancel out V from both sides of equation 17 .The starting point is the discrete-time Bellman equation, equation 3. Rewriting this equation for an instant dt into the future yields

$$\begin{aligned} V_\pi =& u dt + e^{-\rho dt} \mathbb{E}\left[V_\pi\left(s_{t+dt}\right)\right] \\ =& u dt + (1 - \rho dt)\, \mathbb{E}\left[V_\pi + dV_\pi\right] \\ =& V_\pi + dt \cdot \{u - \rho V_\pi + \mathbb{E} dV_\pi\} \\ =& V_\pi + dt \cdot HJB_\pi, \end{aligned} \tag{17}$$

where in the last line the dt terms of order higher than one were discarded. The $\mathbb{E}dV$ term is computed by a direct application of Ito's Lemma:

$$\frac{\mathbb{E}dV}{dt} = \langle \nabla_s V, \mu_s \rangle + \frac{1}{2} trace\left(\sigma_s' \nabla_s^2 V \sigma_s\right)$$

where μ_s and σ_s are the drift and volatility of state vector s, $\nabla_s V$ is the gradient of the value function approximator with respect to the state variables, and $\nabla_s^2 V$ is it's Hessian.

E.3 Proof of Theorem 1

The proof is by induction on the number of shocks, n_S. For $n_S = 1$, the value function drift is:

$$\frac{\mathbb{E}dV}{dt} = \langle \nabla_s V, \mu_s \rangle + \frac{1}{2} \sigma_s' \nabla_s^2 V \sigma_s$$

where σ_s is a $n \times 1$ vector. The first term on the right-hand side of the equation above involves the gradient of the value function, that has computational complexity $O(1) \cdot cost(V)$. The second term involves a Hessian vector product. This product can be compute with 2 runs of the backpropagation algorithm, as follows:

Let $a \equiv grad(V, s)$, where grad denotes differentiation by backpropagation. Let $b \equiv [a(1)s(1), \ldots, a(n)s(n)]$. Then $\nabla_X^2 V = grad(b, X)$. Therefore the computational cost of evaluating the value function drift is $O(1) \cdot cost(V)$.

Suppose now that the computational complexity of the drift is $O(n_S - 1) \cdot cost(V)$ if there are $n_S - 1$ Brownian shocks, and let's analyze the case with n_S shocks. In that case,

$$\begin{array}{rcl} \frac{\mathbb{E}dV}{dt} & = & \langle \nabla_s V, \mu_s \rangle + \frac{1}{2} trace\left(\sigma_s' \nabla_s^2 V \sigma_s\right) \\ & = & \langle \nabla_s V, \mu_s \rangle + \frac{1}{2} \sum_{k=1}^{n_S} \sigma_s^k \nabla_s^2 V \sigma_s^k \end{array} \tag{18}$$

where σ_s^k denote the k^{th} column of the volatility matrix σ_s, that is, $\sigma_s = [\sigma_s^1, \ldots, \sigma_s^{n_S}]$. The first term on the right-hand side of the equation above again has complexity $O(1) \cdot cost(V)$. By linearity of the sum, the quadratic term has complexity $cost(\sum_{k=1}^{S-1} \sigma_s^k \nabla_s^2 V \sigma_s^k)$ + $cost(\sigma_s^{n_S} \nabla_s^2 V \sigma_s^{n_S})$. By the induction hypothesis, the former has complexity $O(n_S - 1) \cdot cost(V)$, while the later has complexity $O(1) \cdot cost(V)$, adding to $O(n_S) \cdot cost(V)$.

[16] See Duffie (2010), for example.

E.4 Proof of Lemma 2 - Q-function in continuous time

The proof is analogous to the proof in E.2.

E.5 Policy evaluation for recursive preferences

The value function of an agent with Epstein-Zin preference satisfies,

$$V = (1 - e^{-\rho dt}) \left((1-\gamma)u\right)^{1/\theta} + e^{-\rho dt} \mathbb{E}\left[V^{1-\gamma}(s')\right]^{1/\theta}.$$

a substitution of the conjectured solution $J = V^{1-\gamma}/(1-\gamma)$ leads to

$$\begin{aligned}
V &= \left\{(1 - e^{-\rho dt}) \left((1-\gamma)u\right)^{1/\theta} + e^{-\rho dt} \mathbb{E}\left[V^{1-\gamma}(s')\right]^{1/\theta}\right\} \\
&\Rightarrow \\
J^{1/\theta} &= \rho dt u^{1/\theta} + (1-\rho dt)(J + \mathbb{E}dJ)^{1/\theta} \\
&= \rho dt u^{1/\theta} + (1-\rho dt) J^{1/\theta}(1 + \tfrac{1}{\theta}\tfrac{\mathbb{E}dJ}{J}) \\
&= J^{1/\theta} + J^{1/\theta} dt \left(\rho \left(\tfrac{u}{J}\right)^{1/\theta} - \rho + \tfrac{1}{\theta}\tfrac{\mathbb{E}dJ}{J}\right) \qquad (19) \\
&\Rightarrow \\
J &= J + dt \left(J\rho\theta \left(\left(\tfrac{u}{J}\right)^{1/\theta} - 1\right) + \tfrac{\mathbb{E}dJ}{dt}\right) \\
&= J + dt \left(f + \tfrac{\mathbb{E}dJ}{dt}\right) \\
&= J + dt \cdot HJB
\end{aligned}$$