Global Identification with Gradient-Based Structural Estimation*

Victor Duarte† Julia Fonseca‡

April 2024

Abstract

This paper develops a gradient-based optimization method to estimate stochastic dynamic models in economics and finance and assess identification globally. By extending the state space to include all model parameters and approximating the mapping between parameters and moments, we only need to solve the model once to structurally estimate parameters. We approximate the mapping between parameters and moments by training a neural network on model-simulated data and then use this mapping to find the set of parameters that minimizes a function of the distance between model and data moments. We show how the mapping between parameters and moments can also be used to assess identification globally, detecting issues that a local diagnostic would miss. We illustrate the algorithm by solving and estimating a dynamic corporate finance model with endogenous investment, costly equity issuance, and capital adjustment costs. In this application, our method reduces the estimation time from many hours to a few minutes.

*We thank Diogo Duarte, Dan Green, Miles Kimball, Leonid Kogan, Jun Li, Adam McCloskey, Karel Mertens, Jonathan Parker, Alessandro Perri, Alex Richter, Dejanir Silva, David Thesmar, Adrien Verdelhan, Toni Whited, Mao Ye, and seminar participants at MIT, the NBER Summer Institute Big Data and High-Performance Computing meeting, the Dallas Fed, Duke Fuqua, the University of Colorado Boulder, and Wharton for helpful comments and discussions.

†Gies College of Business, University of Illinois Urbana-Champaign. Email: vduarte@illinois.edu

‡Gies College of Business, University of Illinois Urbana-Champaign. Email: juliaf@illinois.edu

1 Introduction

Structural estimation is a fundamental tool in many fields of economics and finance, from macroeconomics and industrial organization to asset pricing and corporate finance. A structural estimation procedure typically conducts stochastic dynamic optimization and Monte Carlo simulation to obtain model-implied moments for a given set of parameters in an inner loop and minimizes a function of the distance between model and data moments in an outer loop. This nested problem is computationally intensive for at least two reasons. First, standard approaches solve a dynamic optimization problem for each set of model parameters being evaluated in the outer loop. Moreover, the optimization algorithm used to minimize the distance between the model-implied and data moments is oblivious to the mapping between the model parameters and moments, which precludes the use of gradient-based optimization.

This paper proposes a new approach to structural estimation that circumvents these two bottlenecks. Our method has three key components. First, we train a deep neural network to approximate the value and policy function, augmenting the state space by treating the model parameters as state variables. This makes it unnecessary to solve a complex dynamic programming problem thousands of times. Second, a separate neural network learns the mapping between moments and parameters—which we refer to as the moment function—directly from simulated data. This eliminates the need to compute moments for every given set of parameters and, because the network learns the mapping between parameters and moments, the optimizer in the outer loop can explore this structure to find the model parameters that minimize the corresponding loss function. Specifically, the optimizer has access to analytical gradients and Hessian matrices of the loss function. Third, we use the approximation of the moment function to estimate parameters by minimizing the distance between data and model-implied moments.

In addition to making this efficient estimation algorithm possible, the mapping between moments and parameters serves as the foundation for a global identification diagnostic. The

existence of this mapping makes it feasible to apply a global optimization routine to minimize a function of the distance between model-implied and data moments, which serves as the loss function in our estimation. To assess identification, we minimize this loss function for all parameters but one and assess how it varies for the remaining parameter. This allows us to verify whether a unique global minimum exists and matches our parameter estimate. By optimizing relative to all other parameters rather than holding them fixed at their estimated values, our approach accounts for the fact that perturbing the value of one parameter affects estimates of other parameters. In our application, we show that this approach detects issues that are missed when we hold all other parameters fixed at their estimated values.

We illustrate the method by applying it to a workhorse dynamic corporate finance model with endogenous investment, costly equity issuance, and capital adjustment costs, similar to Gomes (2001) and Hennessy and Whited (2007).

Literature review. The method we develop in this paper is closely related to Simulated Method of Moments (SMM), which approximates model-implied moments using Monte Carlo simulations for each set of parameters being evaluated (McFadden, 1989; Pakes and Pollard, 1989; Lee and Ingram, 1991; Duffie and Singleton, 1993). Our approach differs from SMM in the method used to approximate the model-implied moments. Our method learns a mapping between parameters and moments for an infinite set of parameters directly from simulated observations. This subtle distinction has considerable implications for reducing computing time, as it eliminates the need to solve the model multiple times to evaluate different sets of parameters. This approach is closely related to indirect inference, which uses an auxiliary model to approximate the likelihood (Gourieroux, Monfort, and Renault, 1993). Similarly, Bazdresch, Kahn, and Whited (2017) develop an indirect inference estimation procedure using the empirical policy function as the auxiliary model.

The idea of expanding the set of state variables to include model parameters in order to facilitate estimation was first used in economics by Norets (2012), which proposes a method

using a single-layer neural network in the model solution and Markov chain Monte Carlo for estimation. We contribute to this work by embedding a new solution method using deep neural networks into an estimation procedure that also involves training a separate neural network to approximate the mapping between model parameters and moments.

This work is also related to other papers that study the relationship between parameters and moments to conduct sensitivity analysis. Closest to this paper, Duarte, Duarte, and Silva (2023a) develops a solution method that involves expanding the state space to include model parameters so that, once a solution is obtained, one can easily compute moments for a range of parameter values and assess sensitivity. We contribute to this work by embedding this solution method in an efficient estimation procedure. Specifically, we approximate the mapping between parameters and moments and use it both for estimation and for a global identification diagnostic. We also extend the solution method in Duarte, Duarte, and Silva (2023a) in two key ways. First, we adapt this continuous-time method to a discrete-time setting. Second, we develop an iterative algorithm that focuses on the relevant range of the parameter space as training progresses, allowing us to more quickly obtain an accurate solution for this range.

Andrews, Gentzkow, and Shapiro (2017) proposes a local linear approximation of the mapping between parameters and moments that is used to assess the sensitivity of estimates to misspecification, but not directly in the estimation procedure. Catherine et al. (2023) uses model-implied moments obtained from multiple solutions of a dynamic optimization problem to approximate the moment function, which is then used to estimate parameters and conduct robustness checks. They use their estimate of the moment function to generalize the local approximation of Andrews, Gentzkow, and Shapiro (2017) by computing moments across the full range of possible values for one parameter, while fixing other parameters at their estimated values. Their method differs from traditional methods in that the model is solved multiple times to approximate the moment function instead of at each of the multiple

iteration steps used to estimate parameters in traditional methods.

Our approach differs from these works in two important ways. First, we expand the set of state variables to include all model parameters, altogether eliminating the need to solve the model multiple times at any step of the estimation procedure. Second, our approach allows us to assess identification globally, without holding any parameters fixed, to determine whether the econometric objective function has a unique global minimum. In our application, we show that our global approach can detect issues that are missed when we hold parameters fixed at their estimated values.

This paper is also related to a growing literature using machine learning tools to solve dynamic optimization problems, including Duarte, Duarte, and Silva (2023a), Scheidegger and Bilionis (2019), Maliar, Maliar, and Winant (2021), Azinovic, Gaegauf, and Scheidegger (2022), and Duarte et al. (2023b). We relate to this literature as we also propose a method to solve dynamic optimization problems—which extends the continuous-time method proposed by Duarte, Duarte, and Silva (2023a) to discrete-time problems—and embed it in a fast and efficient estimation procedure. Although we focus on one particular solution method, in principle, our estimation procedure could be extended to incorporate any method capable of handling a large enough number of state variables so that model parameters can be included as states.

Finally, since we apply this method to a dynamic corporate finance model, we also relate to a rich literature that structurally estimates corporate finance models.[1]

The rest of the paper is organized as follows. Section 2 described the main blocks of computations required by our method. Section 3 describes how we combine these blocks of computations into our algorithm. Section 5 applies this method to a dynamic corporate finance model. Section 6 concludes.

[1]See Strebulaev and Whited (2012) for a survey of this literature.

2 Methodology

This section explains our methodology and, in it, we assume knowledge of common machine learning concepts such as neural networks. See Duarte, Duarte, and Silva (2023a) for a brief explanation of these concepts or Sutton and Barto (1998) and Goodfellow, Bengio, and Courville (2016) for a textbook treatment.

Our methodology consists of performing three blocks of computations iteratively. In the first block, taking as input a distribution of model parameters and state variables, we train deep neural networks to approximate the value and policy functions as functions of both states *and* parameters. In the second block, we use the policy function approximation and a distribution of parameters to simulate data. In the third block, we use simulated data to train a neural network to approximate the mapping between model parameters and moments, which we call the moment function. We then use the moment function to estimate model parameters. This section details each of the three blocks and, in Section 3, we describe how our algorithm proceeds iteratively across the three blocks.

2.1 Block 1: Policy iteration

Our procedure for approximating the value and policy functions extends the method of Duarte, Duarte, and Silva (2023a) to discrete-time problems. A key feature of this approach is that, instead of learning the value function for one particular set of parameters, we train a deep neural network to approximate the value function as a function of parameters as well as state variables. This means that we compute value functions for every possible combination of parameters in a given (infinite) set. In machine learning, the value function as a function of parameters and state variables is known as a universal value function.

Expanding the set of state variables to include model parameters will typically lead to a substantial increase in the state space. For instance, in the application of Section 5, we add five parameters to two state variables, increasing the dimension of the state space from

two to seven. Solving this problem thus requires an algorithm that can handle relatively large state spaces, which will generally preclude the use of traditional grid-based methods and polynomial approximations. Note that there are many methods for solving dynamic optimization problems with large state spaces, including ones that use machine learning tools to do so.[2] In principle, any of those algorithms could replace the procedure we describe in this subsection, as long as they can accommodate an expansion of the set of state variables to include model parameters.

Our approach is to use neural networks to approximate the value and policy functions in a policy iteration algorithm that iterates over two steps that separately train the value function and the policy function, respectively, until a pre-specified stopping criterion is met. Policy iteration algorithms start from an arbitrary guess of the policy function and then iterate between (1) estimating the corresponding value function and (2) using the estimated value function to construct a new estimate of the policy function. This class of algorithms typically refers to the first step as policy evaluation and to the second step as policy improvement (e.g. Sutton and Barto, 2018). In what follows, we describe each of the two steps in turn.

2.11 Policy evaluation step

Let u and π denote the utility function and the policy function, respectively. Let V_π denote the value function associated with π. [3] We can write the Bellman equation generally as

$$V_\pi(\Xi) = u(\Xi, \pi(\Xi)) + \beta \mathbb{E} V_\pi(\Xi'), \tag{1}$$

[2]See, for instance, Scheidegger and Bilionis (2019), Maliar, Maliar, and Winant (2021), Azinovic, Gaegauf, and Scheidegger (2022), and Duarte et al. (2023b).

[3]Note that V_π is not what we typically refer to as the value function. The latter is the value function resulting from following the optimal policy function. The former is the expected sum of discounted utility resulting from following a given policy function π. Naturally, the two will be the same if the π is the optimal policy.

where Ξ denotes the expanded state space that includes state variables and model parameters.

We use a deep neural network to parametrize the value function as

$$V_\pi(\Xi) \equiv V_\pi(\Xi; \Theta_V), \tag{2}$$

where Θ_V is the collection of parameters of the neural networks representing the value function. Assume that the vector of next period states Ξ' is a function of the current state Ξ and a vector of shocks $\boldsymbol{\epsilon} \in \mathbb{R}^n$, so that $\Xi' = f(\Xi, \boldsymbol{\epsilon})$. In order to compute the expectation in Eq. (1), we take a 2^{nd} order Taylor expansion around $\boldsymbol{\epsilon} = 0$. Specifically, we use that

$$\mathbb{E}V_\pi(\Xi') = \mathbb{E}V_\pi(f(\Xi, \boldsymbol{\epsilon})) \approx V(f(\Xi, 0)) + \frac{1}{2}\sum_{i=1}^{n} \frac{\partial^2}{\partial {\epsilon_i}^2} V(f(\Xi, 0)) \tag{3}$$

where n is the number of shocks. This is a parallel with the continuous-time method of Duarte, Duarte, and Silva (2023a) that uses that, in continuous time, the approximation given by Eq. (3) is exact.

For a given policy function π, we obtain the implied value function by adjusting network parameters Θ_V to minimize the difference between the left-hand side and the right-hand side of Eq. (1). Specifically, we define the mean squared error as a function of Θ_V as

$$MSE(\Theta_V) = \mathbb{E}\left[\left(V_\pi(\Xi; \Theta_V) - U(\Xi, \pi(\Xi)) - \beta \mathbb{E}V_\pi(\Xi'; \Theta_V)\right)^2\right], \tag{4}$$

approximating $\mathbb{E}V_\pi(\Xi'; \Theta_V)$ using Eq. (3). This is a standard supervised learning problem and can be solved with stochastic gradient descent (i.e. Goodfellow, Bengio, and Courville, 2016). Starting from a random initial guess of the parameter vector Θ_V, a standard (naive) gradient descent algorithm would update the network by changing its parameters in the

direction that most quickly reduces the loss function. The loss function, in this case, is given by the above mean squared error, and one step of gradient descent consists of

$$\Delta\Theta_V = -\alpha\nabla_{\Theta_V} MSE(\Theta_V) = -\alpha\nabla_{\Theta_V}\mathbb{E}\left[(V_\pi(\Xi;\Theta_V) - U(\Xi,\pi(\Xi)) - \beta\mathbb{E}V_\pi(\Xi';\Theta_V))\right], \quad (5)$$

where α is the learning rate. The insight of *stochastic* gradient descent algorithms is to circumvent the computationally costly step of computing the expectation in Eq. (5) by approximating this expectation by drawing an i.i.d. sample of states of size N, with N known in machine learning as the batch size. Note that, since parameters are included as part of the state space, this i.i.d. sample is a sample of both state variables and parameters. We sample from the parameter space by assuming a distribution for parameters, which we describe in more detail in Section 3. We then approximate the mean squared error as

$$\widehat{MSE}(\Theta_V) = \frac{1}{N}\sum_{i=1}^{N}\left(V_\pi(\Xi_i;\Theta_V) - U(\Xi_i,\pi(\Xi)) - \beta\mathbb{E}V_\pi(\Xi_i';\Theta_V)\right)^2, \quad (6)$$

One step of stochastic gradient descent adjusts Θ_V by

$$\Delta\Theta_V = -\alpha\nabla_{\Theta_V}\widehat{MSE}(\Theta_V) \quad (7)$$

We update the network by performing one step of stochastic gradient descent and advance to the policy improvement step.

2.12 Policy improvement step

Given the value function V_π, we would like to obtain a new estimate of the policy function π as

$$\pi(\Xi) = \arg\max_{a} \{u(\Xi, a) + \beta \mathbb{E} V_{\pi}(\Xi')\} \tag{8}$$

This optimization problem is generally computationally costly. Instead of computing the exact maximum to the problem in Eq. (8), we start by using a neural network to parametrize the policy function as

$$\pi(\Xi) \equiv \pi(\Xi; \Theta_{\pi}), \tag{9}$$

where Θ_{π} denotes the collection of parameters of the neural network representing the policy function, and optimize with respect to Θ_{π}.

Plugging the approximation in Eq. (9) into the optimization problem in Eq. (8) yields

$$\pi(\Xi) = \arg\max_{\Theta_{\pi}} \{u(\Xi, \pi(\Xi; \Theta_{\pi})) + \beta \mathbb{E} V_{\pi}(\Xi')\} \tag{10}$$

Instead of performing the maximization above exactly at each iteration, we adjust the parameters vector Θ_{π} gradually using stochastic gradient descent. Specifically, we draw an i.i.d. sample of states and parameters of batch size N and compute the mean of the right-hand side of Eq. (10) across the observations in this sample. To re-frame our optimization as a minimization problem, we define the loss function as the negative of this mean

$$\widehat{L}(\Theta_{\pi}) = -\frac{1}{N} \sum_{i=1}^{N} \left(u(\Xi_i, \pi(\Xi_i; \Theta_{\pi})) + \beta \mathbb{E} V_{\pi}(\Xi_i')\right), \tag{11}$$

where $\mathbb{E} V_{\pi}(\Xi_i')$ is computed using the approximation in Eq. (3). We adjust the parameters of the policy function network by performing one step of stochastic gradient descent.

$$\Delta\Theta_\pi = -\alpha\nabla_\pi \widehat{L}(\Theta_\pi) \tag{12}$$

2.2 Block 2: Simulation

The computations performed in this block are conceptually simple but computationally intensive, which explains why we separate them from the other two blocks. Our goal is to construct a dataset to use in our estimation procedure. To do so, we draw a sample of parameters of size N_d from the parameter distribution. For each vector of parameters in this sample, we use the approximation of the policy function obtained in Block 1 to simulate the model and then compute simulated moments of the vector of variables of interest Y.

2.3 Block 3: Estimation

In this section, we explain how we use the dataset described in Section 2.2 to train a separate neural network to approximate the mapping between model parameters and moments, which we refer to as the moment function. We then describe how we use the moment function for estimation.

Let $\boldsymbol{\beta}$ be the vector of the stacked parameters of a given dynamic model and Y a variable of interest. We are interested in learning the function

$$g(\boldsymbol{\beta}) \equiv \mathbb{E}[Y|\boldsymbol{\beta}]. \tag{13}$$

Note that g satisfies

$$g(\boldsymbol{\beta}) = \arg\min_f \left\{\mathbb{E}\left(Y - f(\boldsymbol{\beta})\right)^2\right\}. \tag{14}$$

We search for a neural network $g(\boldsymbol{\beta}; \Theta_g)$ that approximately solves Eq. (14), that is:

$$\Theta_g^* = \arg\min_{\Theta_g} \left\{ \mathcal{L}(\Theta_g) \right\}, \tag{15}$$

where Θ_g are the parameters of network $g(\boldsymbol{\beta}; \Theta_g)$ and the loss function $\mathcal{L}(\Theta_g)$ is given by

$$\mathcal{L}(\Theta_g) = \mathbb{E}\left(Y - g(\boldsymbol{\beta}; \Theta_g)\right)^2, \tag{16}$$

We draw an i.i.d sample of pairs $\{\beta_i, Y_i\}_{i=1}^N$ of batch size N from the dataset constructed in Block 2, consisting of random draws of the parameter vector $\boldsymbol{\beta}$ and simulated moments of the variable of interest Y. Substituting the population mean with the sample mean, we obtain

$$\Theta_g^* \approx \arg\min_{\Theta_g} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (Y_i - g(\beta_i; \Theta_g))^2 \right\} \tag{17}$$

As before, this optimization can be done with stochastic gradient descent. Once training is complete, the network $g(\boldsymbol{\beta}; \Theta_g)$ has learned a differentiable mapping between parameters and the moments of interest. This has two major implications for structural estimation. First, it is a fast way to evaluate moments for different sets of parameters. Second, we can explore the structure of this mapping to search for parameters using the procedure we describe next.

We use an optimizer to find the global minimum of:

$$d(\boldsymbol{\beta}) \equiv (\hat{g} - g(\boldsymbol{\beta}))' W (\hat{g} - g(\boldsymbol{\beta})) \tag{18}$$

where $\hat{g}$ are the data moments, $g(\boldsymbol{\beta})$ is the output of the moments network, and W is a positive definite weighting matrix. We use as our weighting matrix the inverse of the moment variance-covariance matrix. Following the approach of Erickson and Whited (2002),

we stack the influence functions for all moments and covary them to compute the moment variance-covariance matrix. This choice of weighting matrix has been shown to result in better finite sample performance (Bazdresch, Kahn, and Whited, 2017).

We use a Levenberg-Marquardt algorithm with multiple initial conditions. Levenberg-Marquardt combines gradient descent with a Gauss-Newton algorithm, acting more like gradient descent when far from the solution and more like Gauss-Newton near the solution. Note that it is only possible to use an efficient gradient-based algorithm like Levenberg-Marquardt in this block of computations because we obtain the analytical Jacobian of the moment function from the moment function approximation.

3 Algorithm

In this section, we describe how we combine the two blocks of computations described in Section 2 into our algorithm. This algorithm is asynchronous, as one block of computations does not depend on the completion of the other, and lends itself well to parallelization. We perform all computations on a computer with three NVIDIA V100 Tensor Core GPUs and parallelize this algorithm across the three GPUs. We provide an illustration of the algorithm and this suggested approach to parallelization in Figure 1.

Initialization. We initialize the parameters of the respective networks that approximate the value, policy, and moment functions randomly. To initialize our algorithm, we also need a distribution of parameters and a table containing a sample of size N_r of parameters and state variables, known in machine learning as the replay buffer. We use as the distribution of parameters a sigmoid transformation of a normal distribution.[4] Specifically, for each parameter β^j, we select upper and lower bounds $\underline{\beta}^j$ and $\overline{\beta}^j$ and set

[4]This transformation is similar to the more commonly used beta distribution, but sampling from a normal distribution and applying a sigmoid transformation is less computationally intensive than sampling from a beta distribution.

$$\beta^j = \underline{\beta}^j + (\overline{\beta}^j - \underline{\beta}^j)\boldsymbol{\sigma}(\mu^j + \sigma^j \varepsilon^j), \tag{19}$$

where $\boldsymbol{\sigma}$ is the sigmoid function, μ^j and σ^j are the mean and standard deviation of the normal distribution and $\varepsilon^j \sim \mathcal{N}(0,1)$.

We initialize the distribution of each parameter β^j with $\mu^j = 0$ and $\sigma^j = 1.5$, which results in distributions that approximate a uniform. We sample from these distributions and add this sample of parameters to the initial replay buffer. The sample in the replay buffer is hierarchical, and each observation for the vector of parameters is associated with an observation for the set of state variables. For each vector of parameters in the sample, we select upper and lower bounds for each state variable and sample from a uniform distribution. This hierarchical sampling framework allows us to focus on the relevant region of the state space.

Block 1: Policy Iteration. The policy iteration block of computations is performed continuously by one of the GPUs. This block takes as inputs the distribution of parameters and the replay buffer, and outputs approximations of the policy and value functions as well as a new replay buffer. This block of computations proceeds as follows:

> **Step 1.1:** We start by reading a distribution of parameters. We then obtain a sample of state variables and parameters of batch size N by sampling $0.5N$ from the replay buffer and $0.5N$ from the procedure described in the initialization step. Specifically, we sample from the parameter distribution, select bounds for state variables for each vector of parameters in the sample, and sample state variables from a uniform distribution.
>
> **Step 1.2:** Given a sample of state variables and parameters, we train the neural network that approximates the value function as a function of model parameters and states as described in Section 2.1. We iterate between one step of stochastic gradient descent in the policy evaluation step and one step of stochastic gradient descent in the

policy improvement step for a pre-specified number of iterations, which we denote as an epoch. We replace the old approximations of the policy and value function with the ones produced in this step.

Step 1.3: We update the replay buffer. To do so, we obtain a sample of parameters of size equal to 10% of the replay buffer size, or $0.1N_r$, and use the policy approximation obtained in Step 1.2 to simulate values of state variables for each value of the vector of parameters in this sample. We sample $0.75(0.1N_r)$ from the parameter distribution and $0.25(0.1N_r)$ from the initial distribution with mean zero and standard deviation 1.5, which approximates a uniform distribution. We discard 10% of the oldest observations in the replay buffer and replace it with the sample obtained in this step. We then return to step 1.1 and proceed iteratively until a criterion is met.

Block 2: Simulation. The simulation block of computations is performed continuously in a second GPU. This block takes as inputs the distribution of parameters and a neural network approximating the policy function, and outputs a dataset with random draws of the parameter vector and simulated moments of the variable of interest. This block of computations proceeds as follows:

Step 2.1: Given the parameter distribution, we draw an i.i.d. sample of parameters of size N_d.

Step 2.2: For each vector of parameters in this sample, we use the approximation of the policy function to simulate the model and then compute simulated moments of the vector of variables of interest Y, producing a dataset with parameter vectors and simulated moments. We then return to step 2.1 and proceed iteratively until a criterion is met.

Block 3: Estimation. The estimation block of computations is performed in a third GPU. This block takes as inputs a dataset with random draws of the parameter vector and

simulated moments of the variable of interest and outputs new distributions of parameters and a neural network approximating the moment function. This block of computations proceeds as follows:

Step 3.1: We draw a small i.i.d. sample of parameters and states of batch size N from the dataset constructed in Block 2.

Step 3.2: Given a sample of parameters and states, we train the neural network that approximates the moment function as described in Section 2.3 for one epoch.

Step 3.3: To update the standard deviation of the parameter distribution in Step 3.5 below, we keep a total of four versions of the moment function: the moment function obtained in step 3.2 in the current iteration plus the moment functions from the three previous iterations. We use this as our approach to obtain numerical standard errors because it reduces the value of standard errors, and thus samples from a narrower range of the parameter distribution, as the solution converges and all versions of the moment function approximation become increasingly similar. An alternative that is simpler to implement is to anneal the standard deviation of the parameter distribution to a small number during training.

Step 3.4: For each moment function approximation, we define the distance function given by Eq. (18). We then use a Levenberg-Marquardt optimization routine with multiple initial conditions to find the vector of parameters that minimizes the distance function as described in Section 2.3 for one epoch. This produces a total of four estimates of the vector of parameters, one for each moment function approximation.

Step 3.5: To ensure that our algorithm focuses on the relevant subset of the parameter space, we update the mean and standard deviation of the parameter distribution. For each parameter, we replace the mean and standard deviation with the mean and standard deviation of the four parameter estimates obtained in Step 3.4. Again, an

alternative for researchers who prefer to have a single moment function and skip step 3.3 is to replace the mean with a single parameter estimate and anneal the standard deviation during training. After updating the mean and standard deviation, we return to step 3.1 and proceed iteratively until a criterion is met.

4 Global identification diagnostic

In addition to producing a model solution and parameter estimates, the algorithm in Section 3 produces a mapping between moment and parameters. With this moment function on hand, it is as if we have the mapping between parameters and moments in closed form. It is then simple and computationally cheap to assess how the distance function from Eq. (18) varies with each parameter.

To evaluate if parameter β^j is identified, we minimize the distance function in Eq. (18) across all *other* parameters to obtain a "minimum loss function" as a function of β^j, $L(\beta^j)$. By minimizing the distance function with respect to all other parameters instead of fixing them at their estimated values, this procedure accounts for the fact that changing the value of one parameter can change the optimal value of other parameters. Specifically, we solve

$$L(\beta^j) = \min_{\boldsymbol{\beta}^{-j}} \; d(\beta^j, \boldsymbol{\beta}^{-j}) \equiv \left(\hat{g} - g(\beta^j, \boldsymbol{\beta}^{-j})\right)' W \left(\hat{g} - g(\beta^j, \boldsymbol{\beta}^{-j})\right), \tag{20}$$

where $\boldsymbol{\beta}^{-j}$ denotes the vector of all parameters other than β^j. We approximate $g(\boldsymbol{\beta})$ using the network $g(\boldsymbol{\beta}; \Theta_g)$ obtained using the algorithm in Section 3 above. As before, we use the inverse of the moment variance-covariance matrix as our weighting matrix and solve the minimization problem in Eq. 20 using the Levenberg-Marquardt algorithm with multiple initial conditions.

We then plot the minimum loss function $L(\beta^j)$ as a function of β^j and check whether it has a unique global minimum and, if so, we conclude that β^j is identified by the set of moments used in the estimation procedure. We show examples of this approach to assessing identification in Section 5.

5 Application: A dynamic corporate finance model

In this section, we apply our method to a workhorse corporate finance model similar to Gomes (2001) and Hennessy and Whited (2007). After describing the model environment, we start by showing that our method can recover known parameters and illustrating our global identification diagnostic. Next, we estimate the model using Compustat data and compare the performance of our method with the more common approach of combining value function iteration with a gradient-free global optimization method. Finally, we show that our global identification diagnostic can detect a limitation in our estimation with Compustat data that a local diagnostic would miss.

5.1 Model description

Time is discrete. An infinitely-lived firm chooses investment to maximize the expected present value of distributions to shareholders and is subject to investment adjustment costs and costly equity issuance. The firm uses capital in a decreasing-returns technology that generates operating income Y_t according to

$$Y_t = z_t K_t^{\alpha}, \tag{21}$$

where $\alpha \in (0, 1)$ governs the degree of returns to scale, z_t is a productivity shock, and K_t is the stock of capital. The productivity shock z_t is log-normally distributed and follows

$$\log(z_t) = \rho \log(z_{t-1}) + \varepsilon_t, \text{ where } \varepsilon_t \sim \mathcal{N}(0, \sigma_z) \tag{22}$$

The firm chooses investment I_t each period and capital accumulation follows

$$K_{t+1} = (1 - \delta)K_t + I_t, \tag{23}$$

where δ is the depreciation rate. Cash flows $E(K_t, K_{t+1}, z_t)$ are given by

$$E(K_t, K_{t+1}, z_t) = z_t K_t^{\alpha} - K_{t+1} + (1 - \delta)K_t - \frac{\chi I_t^2}{2} \tag{24}$$

The term $\frac{\chi I_t^2}{2}$ introduces a convex capital adjustment cost. Positive cash flows are distributed to shareholders while negative cash flows imply that the firm issues equity, which is subject to a linear cost $\lambda > 0$. Distributions to shareholders $D(K_t, K_{t+1}, z_t)$ are thus given by

$$D(K_t, K_{t+1}, z_t) = \begin{cases} E(K_t, K_{t+1}, z_t), & \text{if } E(K_t, K_{t+1}, z_t) \geq 0 \\ (1 + \lambda)E(K_t, K_{t+1}, z_t), & \text{if } E(K_t, K_{t+1}, z_t) < 0 \end{cases} \tag{25}$$

The firm solves

$$V(K_t, z_t) = \max_{K_{t+1}} \{D(K_t, K_{t+1}, z_t) + \beta \mathbb{E} V(K_{t+1}, z_{t+1})\} \tag{26}$$

subject to Eq. (25) and the law of motion of the productivity shock (Eq. (22)) and capital (Eq. (23)).

5.2 Recovering known parameters

To assess the performance of our method, we start by showing the results of an exercise in which the true parameter values are known. To do so, we solve the model using value function iteration for a given set of parameters, use this solution to produce simulated data, use simulated data to compute moments, and then use the algorithm described in Section 3 to solve and estimate the model targeting these moments. The advantage of this exercise is that we know the true parameters that generated the underlying data and thus can assess the ability of our method to recover the correct parameters.

We repeat this exercise multiple times to ensure that our method performs well across the parameter space. We draw 1,000 parameter vectors from a uniform distribution and discard those in which the implied equity issuance is lower than 0.01.[5] For each parameter vector in this set, we solve the model using value function iteration, simulate data using the policy functions, compute the above moments in each of the model-generated datasets, and estimate the model using our method.

We estimate five parameters: the rate of depreciation δ, the curvature of the profit function α, the cost of equity issuance λ, the auto-correlation of the productivity shock ρ, and the innovation of the productivity shock process σ_z. We target five moments: the average investment rate $\frac{I_t}{K_t}$, the average profitability rate $\frac{Y_t}{K_t}$, the average equity issuance $\frac{\min\{E(K_t,K_{t+1},z_t),0\}}{K_t}$, the auto-correlation of $\frac{Y_t}{K_t}$ (computed by regressing $\frac{Y_t}{K_t}$ on $\frac{Y_{t-1}}{K_{t-1}}$), and the standard deviation of the residuals of this regression. We fix the values of the remaining parameters, setting β to 0.98 and the capital adjustment cost χ to 0.03.

We choose the following bounds for each of the five parameters: $\delta \in [0.02,\ 0.12]$, $\alpha \in [0.3,\ 0.6]$, $\lambda \in [0.0,\ 0.3]$, $\rho \in [0.5,\ 0.9]$, $\sigma_z \in [0.05,\ 0.6]$. As we describe in Section 3, the sample in the replay buffer is hierarchical, meaning that each observation for the vector of

[5]For very low levels of equity issuance, the equity issuance cost is not identified.

parameters is associated with an observation for the set of state variables. For each vector of parameters the sample of parameters, we select upper and lower bounds for productivity using a Tauchen approximation as

$$\log \overline{z} = 4\frac{\sigma_z}{1-\rho^2} \tag{27}$$

$$\log \underline{z} = -4\frac{\sigma_z}{1-\rho^2} \tag{28}$$

To set bounds for capital, we start by computing the steady state level of capital as

$$K_{ss} = \left(\frac{\alpha\beta}{1-(1-\delta)\beta}\right)^{\frac{1}{1-\alpha}} \tag{29}$$

We then define the upper and lower bound for capital as

$$\log \overline{K} = \log K_{ss} + \overline{z} \tag{30}$$

$$\log \underline{K} = \log K_{ss} + \underline{z} \tag{31}$$

We start by showing that the moment function we estimate is an accurate representation of the true moments. In Fig. 2, we show scatter plots of data moments and moments implied by the moment function, with each panel representing a different moment. The x-axis of each panel represents the data moments implied by each of the parameter vectors that we randomly draw and the y-axis represents the moments obtained by applying the moment network to the same set of parameter vectors. We also report the 45-degree line in solid black so that, if the fit between data moments and those implied by the moment function were perfect, all dots should lie on that line. We can see that the fit is remarkably good,

with an R^2 of 1 across all parameters.

The close fit of the moment function is important since errors in moment conditions would pose challenges for estimating parameters. Fortunately, neural networks are flexible function approximators and can approximate even non-linear functions. Moreover, the iterative algorithm of Section 3 focuses attention on the relevant parameter range as training progresses, which also leads to a better approximation of the moment function in that range.

Next, we show that we also obtain a close fit of the true parameters. In Fig. 3, we show scatter plots of the true parameters and the parameters obtained in our estimation, with each panel representing a different parameter. In each panel, the x-axis is the set of randomly drawn parameters and the y-axis is the set of parameter estimates, with the 45-degree line shown in black. Our parameter estimates are also a close fit to the true parameters, with R^2s ranging from 0.99 to 1.

5.3 Minimum loss functions: Assessing identification

Next, we show how to use the mapping between parameters and moments to assess identification globally. After solving and estimating the model for each randomly drawn vector of true parameters in Section 5.2, we use a Levenberg-Marquardt algorithm with multiple initial conditions to minimize the loss function given by Eq. (18), the distance between model and data moments, for all parameters but one. This gives us the global minimum of the loss function as a function of a parameter, allowing us to check whether a global minimum exists. If there is not a unique global minimum, this suggests that the parameter is not identified by this set of moments. We repeat this exercise for all five parameters.

We illustrate the usefulness of this approach by showing that the parameters in this model are identified by this set of moments for some true parameter values but not for others. In Fig. 4, we plot the minimum loss function as a function of productivity innovation σ_z (Panels 4a and 4c) and equity issuance cost λ (Panels 4b and 4d) for two different

true parameter vectors. In version 1 (Panels 4a and 4b), the true parameter vector is $[\alpha,\ \delta,\ \lambda,\ \rho,\ \sigma_z] = [0.484,\ 0.110,\ 0.040,\ 0.635,\ 0.370]$. In version 2 (Panels 4c and 4d), the true parameter vector is $[\alpha,\ \delta,\ \lambda,\ \rho,\ \sigma_z] = [0.362,\ 0.106,\ 0.095,\ 0.846,\ 0.140]$. The red vertical line denotes the true parameter value.

In this model, σ_z seems to be identified by this set of moments across the true parameter space. We illustrate this in Panel 4a and 4c, which shows that the minimum loss function for σ_z has a global minimum for the two different vectors of true parameters.

In contrast, λ is identified by this set of moments for some vectors of true parameters, but not for others. In Panel 4b, the minimum loss function has a unique global minimum, suggesting λ is identified by this set of moments when the true parameter vector is $[\alpha,\ \delta,\ \lambda,\ \rho,\ \sigma_z] = [0.484,\ 0.110,\ 0.040,\ 0.635,\ 0.370]$. However, the same is not true in Panel 4d, when the true parameter vector is $[\alpha,\ \delta,\ \lambda,\ \rho,\ \sigma_z] = [0.362,\ 0.106,\ 0.095,\ 0.846,\ 0.140]$. In this case, the minimum loss function is essentially flat, suggesting λ is not identified.

5.4 Estimation using Compustat data

Finally, we use data from Compustat between 1970 and 2019 to construct data moments and use the algorithm of Section 3 to estimate the model targeting these moments.

5.41 Data construction

We exclude firms in the financial (SIC code 6) and regulated sectors (SIC code 49), as well as quasi-governmental firms (SIC code 9). We exclude all observations that have missing data in any of the variables that we use in our analysis. We also exclude observations in which total assets are less than 10 million dollars, in which sales or assets increase more than 200%, and those with negative sales. Finally, we exclude firms that have fewer than four consecutive observations.

We construct variables as:

1. investment-to-asset ratio = $\frac{capx}{l.at}$
2. profit-to-asset ratio = $\frac{oibdp}{l.at}$
3. equity-issuance-to-asset ratio (computed net of repurchases) = $\frac{sstk-prstkc}{l.at}$

We winsorize all ratios at the 1st and 99th percentiles. Because firms in this model are ex-ante homogeneous, we remove firm fixed effects from these variables by subtracting the within-firm average. Removing firm fixed effects implies that all variables will have a mean of zero, and so we add back the sample average. To compute the autocorrelation of the profit-to-asset ratio, we regress the profit-to-asset ratio on its lag and include year fixed effects. As a separate targeted moment, we also compute the standard deviation of the residuals of this regression.

5.42 Estimation results

We estimate the model using the algorithm described in Section 3. We start by illustrating our endogenous sampling framework for two parameters in Fig. 5. The y-axis corresponds to values of the equity issuance cost λ and the x-axis has values of the productivity innovation σ_z. Each panel shows a different sample and each blue dot corresponds to an observation in the sample, with estimated parameter values shown in orange. Panel 5a shows the sample from uniform distributions that we obtain during the initialization step described in Section 3. Panel 5b shows the final sample that we obtain after running the algorithm for 15 minutes. As we describe in Section 3, our algorithm updates the mean and standard deviation of the parameter vector during training and samples 75% of observations from that distribution. This allows us to quickly obtain a high-quality solution in the region of the parameter space around the parameter estimates.

Next, we show the results of our estimation in Tables 1 and 2. As shown in Table 1, our method does a very good job of matching the data moments. Table 2 reports parameter estimates, with standard errors in parentheses computed following the influence function

approach of Erickson and Whited (2002). We obtain precise estimates of all parameters.

We also compare our algorithm against the more standard approach of solving the model using value function iteration and estimating parameters with a gradient-free global optimization method—differential evolution—targeting the same data moments. We report the estimates obtained with this approach in the last row of Table 2. For all parameters but one, the estimate we obtain with the standard approach falls within the 95% confidence interval for our estimate. The one exception is α, the curvature of the profit function, for which the estimate we obtain with the standard approach is just outside the confidence interval for our method. As we discuss in Section 5.43 below, this is potentially a reflection of our choice of moments, which do not identify this parameter as well as others.

In Fig. 6, we also report parameter estimates over the run time using our method and the standard approach. Each panel refers to a different parameter and, across all panels, the x-axis is run time in minutes and the y-axis shows estimated parameter values obtained with our method (blue line) and value function iteration plus differential evolution (orange line). For each method, we run the algorithm five times with different seeds and different random initial conditions. The solid lines show the average parameter estimate over time for each method and the bands correspond to plus or minus one standard deviation. We run our method for 30 minutes and differential evolution for 10 hours. For all parameters, we can see that our algorithm converges in a very small fraction of the time that it takes the standard approach to reach a similar estimate.

5.43 Global identification

Next, we illustrate our global identification diagnostic in the context of the estimation using Compustat data and show that it can detect a limitation that a local diagnostic would miss. As in Section 5.3 we start by solving and estimating the model targeting the data moments and then use a Levenberg-Marquardt algorithm with multiple initial conditions to minimize

the loss function given by Eq. (18), the distance between model and data moments, for all parameters but one. We show minimum loss functions for all five parameters in Fig. 7. In panel 7a, we see that the minimum loss function is relatively flat for α, the curvature of the profit function. This is a likely explanation for why we obtain larger differences in estimates of this parameter across different estimation procedures than in other parameters. The change in the minimum loss function from going from our estimate of 0.428 to the 0.423 estimate we obtain with differential evolution (Table 2) is of the order of $1e^{-7}$, suggesting that even very small numerical error would be sufficient to confound estimates within this range.

Finally, we use this example to demonstrate the usefulness of a global diagnostic that minimizes the distance between data and model-implied moments for all parameters but one by comparing these results with the more common approach of holding all parameter values but one fixed at their estimated values. Fig. 8 shows the same minimum loss functions as in Fig. 7 in blue and, in orange, shows the distance between data and model-implied moments when we vary the parameter denoted in the figure caption and hold all other parameters fixed at their estimated value. Based on this local approach, we would conclude that all parameters are identified by this set of moments and would not detect the challenge to identifying the curvature of the profit function.

6 Conclusion

We propose a gradient-based optimization method to efficiently estimate stochastic dynamic models by expanding the set of state variables to include all model parameters and approximating the mapping between parameters and model-implied moments directly from simulated observations. After a moment network is trained on a data set composed of simulated observations, optimization can be carried out as if we had the estimating equations in closed form. Essentially, we propose a way to make SMM as tractable as GMM. We also

show how to use moment networks to globally assess identification by computing global loss functions.

We illustrate our approach by solving and estimating a dynamic corporate finance model with endogenous investment, costly equity issuance, and capital adjustment costs. The expanded state space includes two state variables and five model parameters. We solve the model for an infinite set of parameters and obtain precise parameter estimates in 15 minutes. Finally, we show that our global identification diagnostic can detect issues that local diagnostics would miss.

References

Andrews, Isaiah, Matthew Gentzkow, and Jesse M. Shapiro. Measuring the sensitivity of parameter estimates to estimation moments. *The Quarterly Journal of Economics*, 132 (4):1553–1592, 06 2017. ISSN 0033-5533. doi: 10.1093/qje/qjx023. URL https://doi.org/10.1093/qje/qjx023.

Azinovic, Marlon, Luca Gaegauf, and Simon Scheidegger. Deep equilibrium nets. *International Economic Review*, 63(4):1471–1525, 2022. doi: https://doi.org/10.1111/iere.12575. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/iere.12575.

Bazdresch, Santiago, R. Jay Kahn, and Toni M. Whited. Estimating and testing dynamic corporate finance models. *The Review of Financial Studies*, 31(1):322–361, 07 2017. ISSN 0893-9454. doi: 10.1093/rfs/hhx080. URL https://doi.org/10.1093/rfs/hhx080.

Catherine, Sylvain, Mehran Ebrahimian, David Sraer, and David Thesmar. Robustness checks in structural analysis. Technical report, Working paper, 2023.

Duarte, Victor, Diogo Duarte, and Dejanir Silva. Machine learning for continuous-time finance. *SSRN*, 2023a. URL https://ssrn.com/abstract=3012602.

Duarte, Victor, Julia Fonseca, Aaron Goodman, and Jonathan Parker. Simple allocation rules and optimal portfolio choice over the lifecycle. Technical report, Working paper, 2023b.

Duffie, Darrell, and Kenneth J. Singleton. Simulated moments estimation of markov models of asset prices. *Econometrica*, 61(4):929–952, 1993. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/2951768.

Erickson, Timothy, and Toni M. Whited. Two-step GMM estimation of the errors-in-variables model using high-order moments. *Econometric Theory*, 18(3):776–799, 2002. ISSN 02664666, 14694360. URL http://www.jstor.org/stable/3533649.

Gomes, Joao F. Financing investment. *American Economic Review*, 91(5):1263–1285, December 2001. doi: 10.1257/aer.91.5.1263. URL https://www.aeaweb.org/articles?id=10.1257/aer.91.5.1263.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Gourieroux, C., A. Monfort, and E. Renault. Indirect inference. *Journal of Applied Econometrics*, 8:S85–S118, 1993. ISSN 08837252, 10991255. URL http://www.jstor.org/stable/2285076.

Hennessy, Christopher A., and Toni M. Whited. How costly is external financing? evidence from a structural estimation. *The Journal of Finance*, 62(4):1705–1745, 2007. ISSN 00221082, 15406261. URL http://www.jstor.org/stable/4622315.

Lee, Bong-Soo, and Beth Ingram. Simulation estimation of time-series models. *Journal of Econometrics*, 47(2-3):197–205, 1991. URL https://EconPapers.repec.org/RePEc:eee:econom:v:47:y:1991:i:2-3:p:197-205.

Maliar, Lilia, Serguei Maliar, and Pablo Winant. Deep learning for solving dynamic economic models. *Journal of Monetary Economics*, 122:76–101, 2021. ISSN 0304-3932. doi: https://doi.org/10.1016/j.jmoneco.2021.07.004. URL https://www.sciencedirect.com/science/article/pii/S0304393221000799.

McFadden, Daniel. A method of simulated moments for estimation of discrete response models without numerical integration. *Econometrica*, 57(5):995–1026, 1989. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1913621.

Norets, Andriy. Estimation of dynamic discrete choice models using artificial neural network approximations. *Econometric Reviews*, 31(1):84–106, 2012. doi: 10.1080/07474938.2011.607089. URL http://dx.doi.org/10.1080/07474938.2011.607089.

Pakes, Ariel, and David Pollard. Simulation and the asymptotics of optimization estimators.

Econometrica, 57(5):1027–1057, 1989. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1913622.

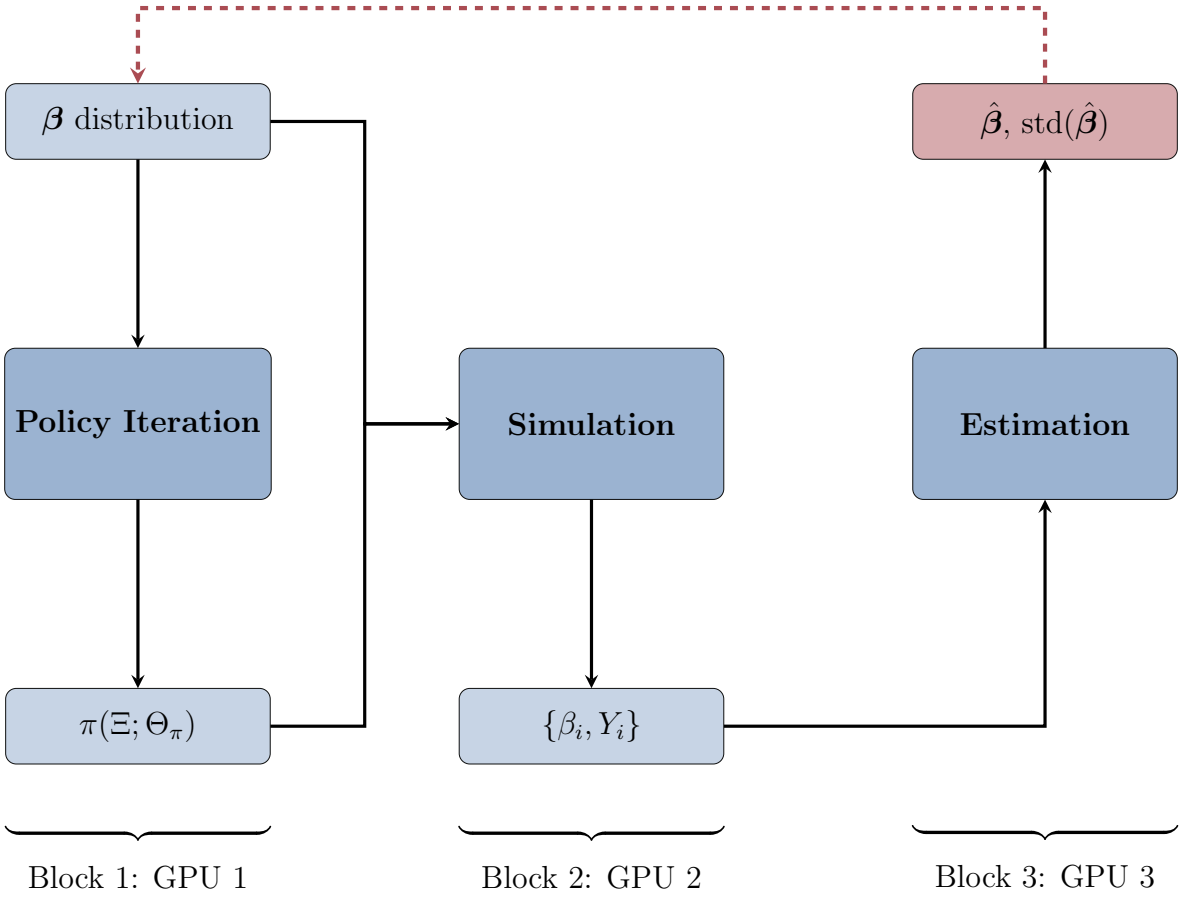
Scheidegger, Simon, and Ilias Bilionis. Machine learning for high-dimensional dynamic stochastic economies. *Journal of Computational Science*, 33:68–82, 2019. ISSN 1877-7503. doi: https://doi.org/10.1016/j.jocs.2019.03.004. URL https://www.sciencedirect.com/science/article/pii/S1877750318306161.

Strebulaev, Ilya A., and Toni Whited. Dynamic models and structural estimation in corporate finance. *Foundations and Trends(R) in Finance*, 6(1–2):1–163, 2012. URL https://EconPapers.repec.org/RePEc:now:fntfin:0500000035.

Sutton, Richard S., and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

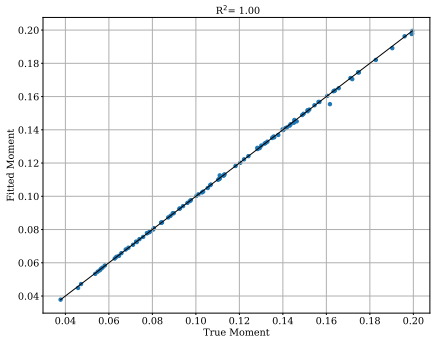
Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Figure 1: Illustration of Algorithm

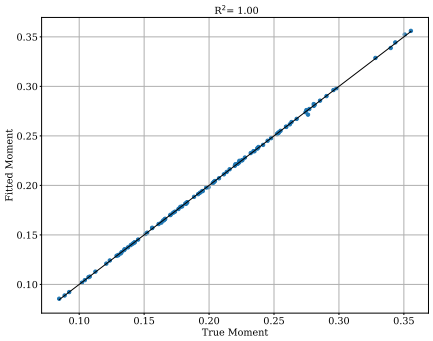


This figure illustrates the algorithm presented in Section 3. Computations are described in Section 2. GPU 1, 2, and 3 refer to three NVIDIA V100 Tensor Core GPUs.

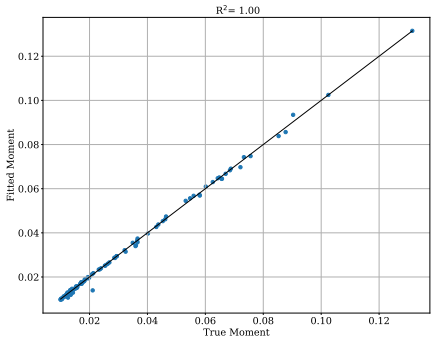
Figure 2: True vs. Fitted Moments



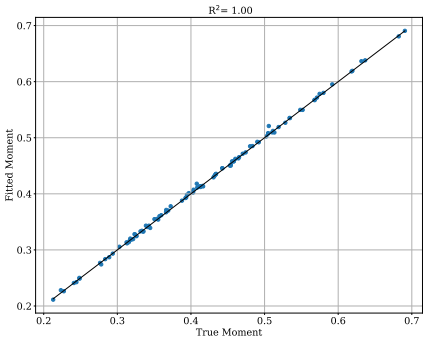
(a) Average investment rate



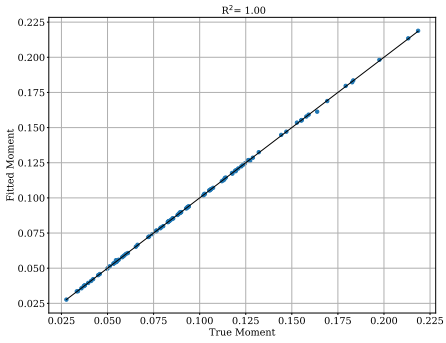
(b) Average profitability rate



(c) Average equity issuance



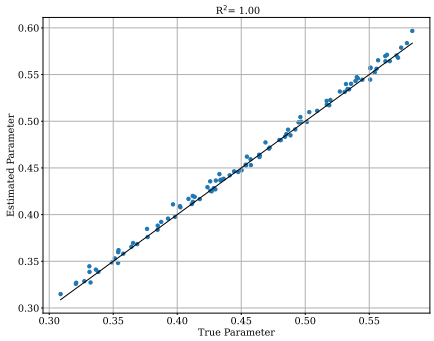
(d) Auto-correlation of profitability



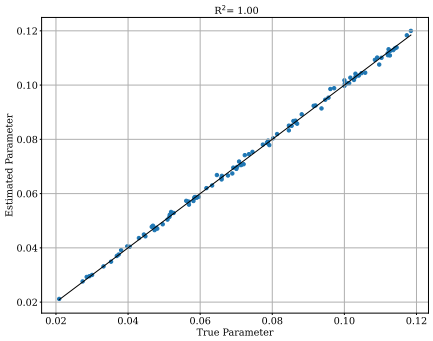
(e) Standard deviation of profitability residuals

This figure shows scatter plots of data moments and moments implied by the moment function. We draw 1,000 parameter vectors from a uniform distribution and discard those in which the implied equity issuance is lower than 0.01. The x-axis of each panel represents the data moments implied by the true parameter vectors and the y-axis represents the moments obtained by applying the moment network to the same set of parameter vectors. The run time for each parameter vector is 15 minutes. We set the batch size to 512, the replay buffer size to 50,000, the dataset size to 5,000, the learning rate to $1e^{-3}$, and an epoch to 100 iterations. Our neural networks have 2 hidden layers, each with 256 nodes. We use a Sigmoid Linear Unit (SiLU) activation function, defined as $silu(x) = x\sigma(\mathbf{x})$, where σ is the sigmoid function.

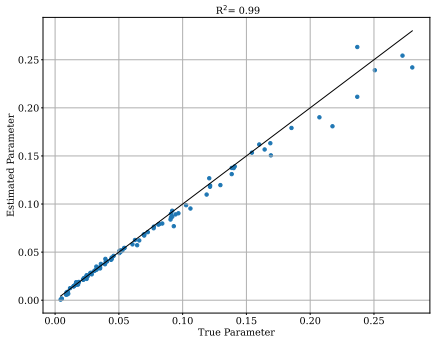
Figure 3: True vs. Estimated Parameters



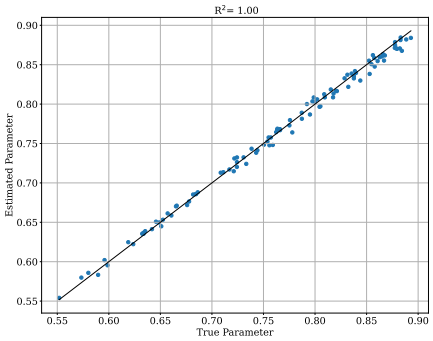
(a) Curvature of profit function (α)



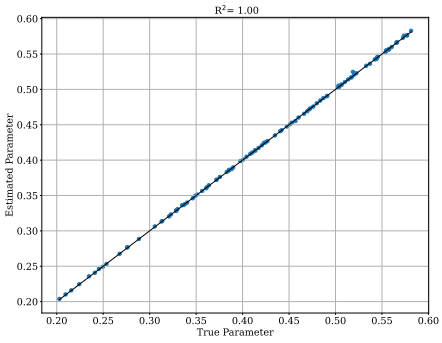
(b) Depreciation rate (δ)



(c) Equity issuance cost (λ)



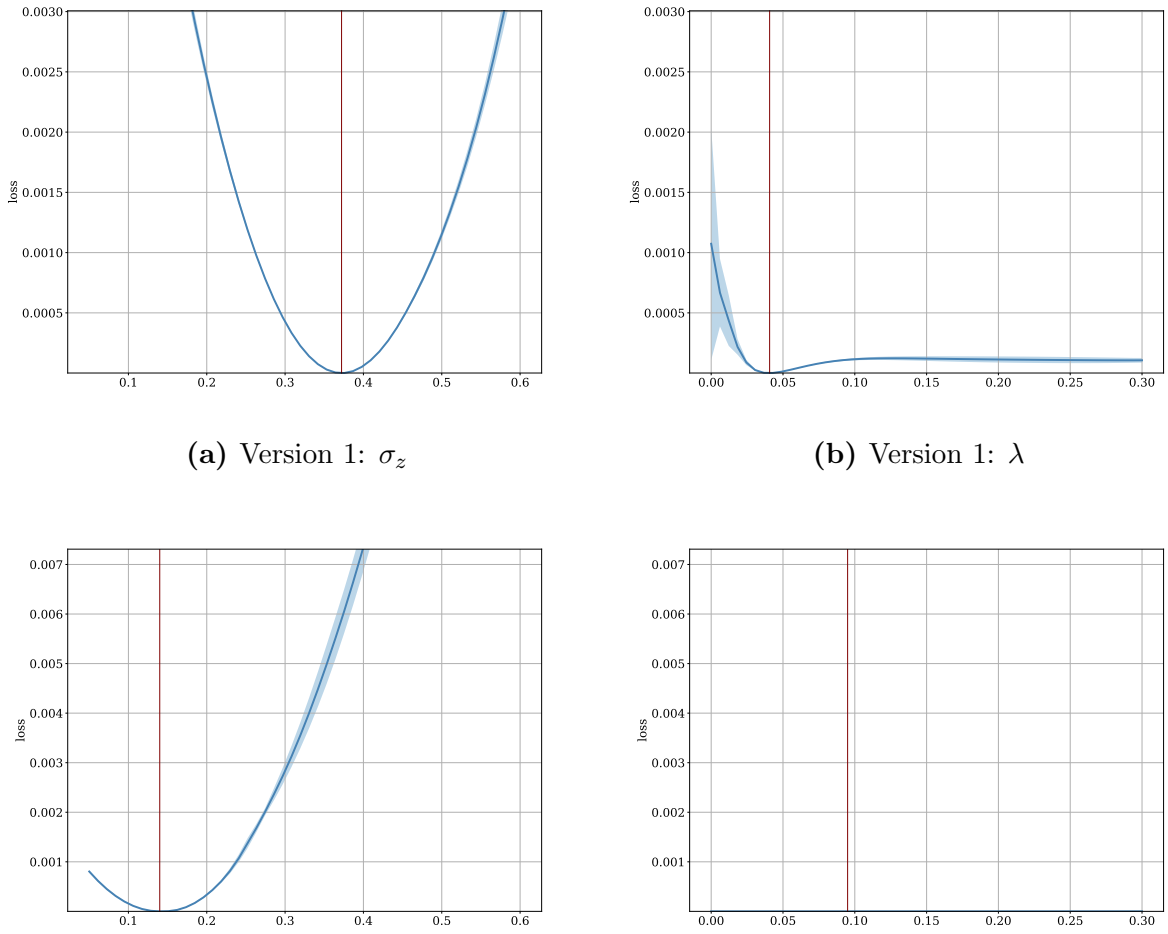
(d) Auto-corr. of productivity (ρ)



(e) Productivity innovation (σ_z)

This figure shows scatter plots of true and estimated parameters. We draw 1,000 parameter vectors from a uniform distribution and discard those in which the implied equity issuance is lower than 0.01. The x-axis of each panel shows values of the true parameter and the y-axis represents parameter estimates obtained using our method when targeting the moments implied by each true parameter vector. The solid black line is the 45-degree line. The run time for each parameter vector is 15 minutes. We set the batch size to 512, the replay buffer size to 50,000, the dataset size to 5,000, the learning rate to $1e^{-3}$, and an epoch to 100 iterations. Our neural networks have 2 hidden layers, each with 256 nodes. We use a Sigmoid Linear Unit (SiLU) activation function, defined as $silu(x) = x\sigma(\mathbf{x})$, where σ is the sigmoid function.

Figure 4: Minimum Loss Function: Known Parameters



(a) Version 1: σ_z

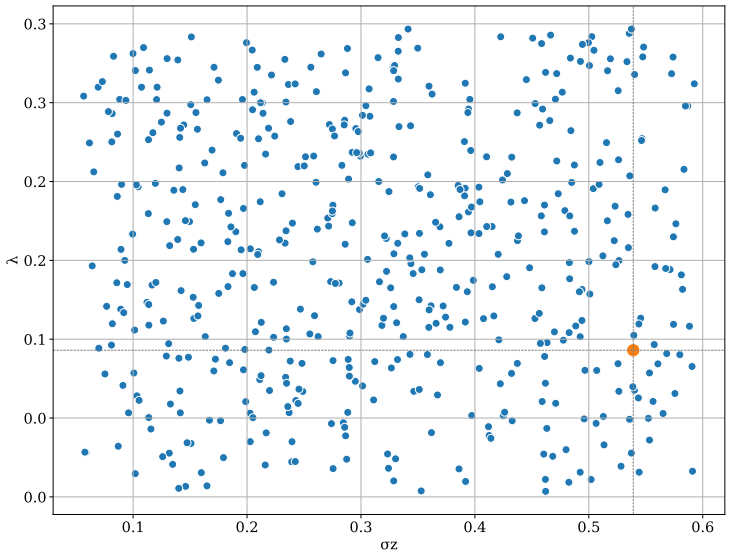
(b) Version 1: λ

(c) Version 2: σ_z

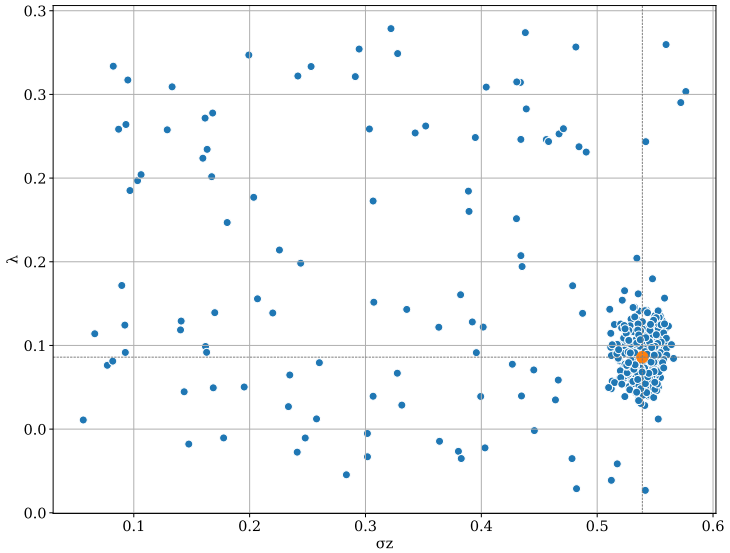
(d) Version 2: λ

This figure shows the minimum loss function as a function of productivity innovation σ_z (Panels 4a and 4c) and equity issuance cost λ (Panels 4b and 4d) for two different true parameter vectors. In version 1 (Panels 4a and 4b), the true parameter vector is $[\alpha,\ \delta,\ \lambda,\ \rho,\ \sigma_z] = [0.484,\ 0.110,\ 0.040,\ 0.635,\ 0.370]$. In version 2 (Panels 4c and 4d), the true parameter vector is $[\alpha,\ \delta,\ \lambda,\ \rho,\ \sigma_z] = [0.362,\ 0.106,\ 0.095,\ 0.846,\ 0.140]$. In each panel, we use a global optimization routine to minimize the loss function given by Eq. (18), the distance between model and data moments, for all parameters other than the one denoted in the caption. We compute the minimum loss function for the network parameters of the last ten iterations and compute the mean and standard deviation across these ten versions. We show two standard deviation bands along with means. The red vertical line denotes the true parameter value. The run time for each parameter vector is 15 minutes. We set the batch size to 512, the replay buffer size to 50,000, the dataset size to 5,000, the learning rate to $1e^{-3}$, and an epoch to 100 iterations. Our neural networks have 2 hidden layers, each with 256 nodes. We use a Sigmoid Linear Unit (SiLU) activation function, defined as $silu(x) = x\sigma(\mathbf{x})$, where σ is the sigmoid function.

Figure 5: Endogenous sampling



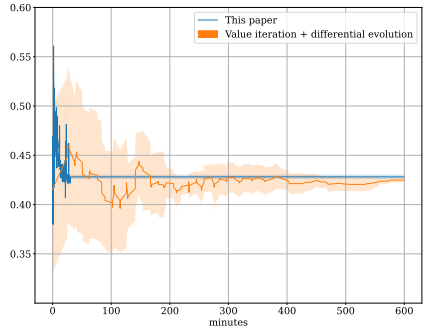
(a) Initial sample



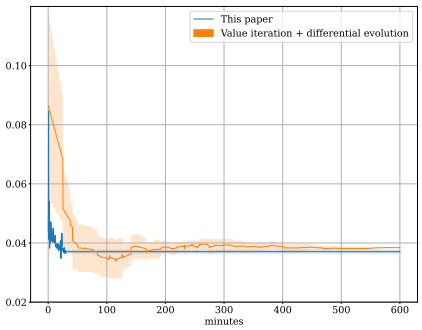
(b) Final sample

This figure illustrates our endogenous sampling method for two parameters, the equity issuance cost (λ) and the productivity innovation (σ_z). Each blue dot is an observation in the sample and the estimated parameters are shown in orange. Panel 5a shows the initial sample, described in the initialization step of the algorithm in Section 3. Panel 5b shows the sample after running the algorithm for 15 minutes.

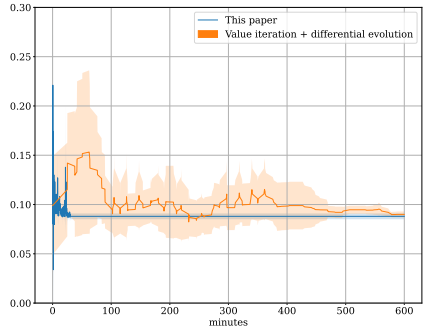
Figure 6: Time to Solution



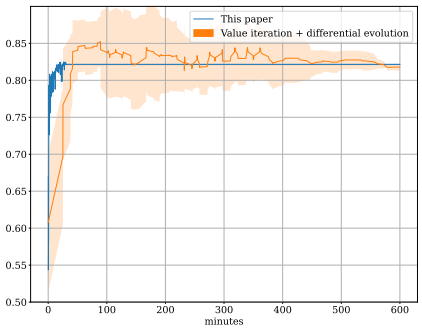
(a) Curvature of profit function (α)



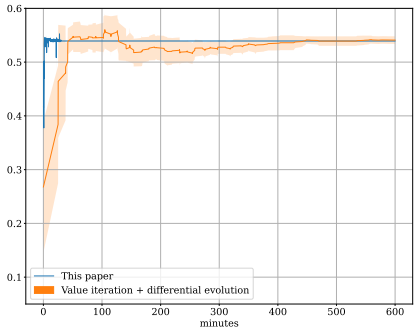
(b) Depreciation rate (δ)



(c) Equity issuance cost (λ)



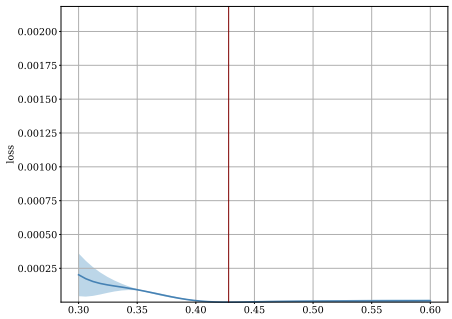
(d) Auto-corr. of productivity (ρ)



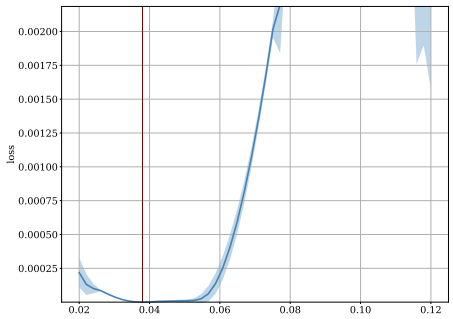
(e) Productivity innovation (σ_z)

This figure shows estimated parameter values on the y-axis and run time in minutes on the x-axis, for both our estimation method (blue line) and the standard approach of solving the model with value function iteration and estimating parameters using differential evolution (orange line). Each point in either the blue or the orange line is the average over five runs with different seeds and different random initial conditions. We plot the average parameter value as well as bands of plus or minus one standard deviation. We run our method for 30 minutes and differential evolution for 10 hours.

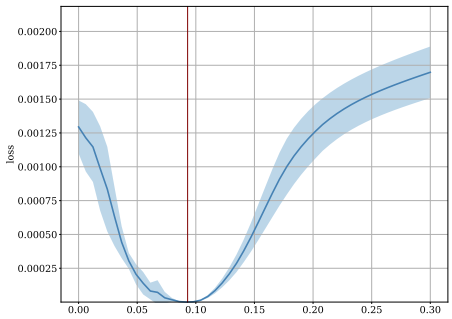
Figure 7: Minimum Loss Function: Data



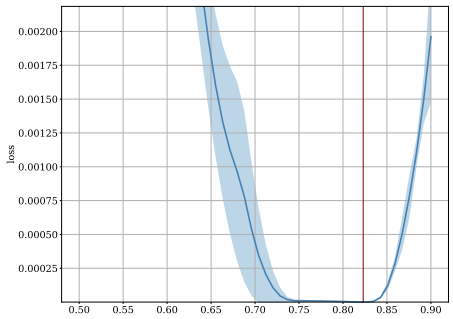
(a) Curvature of profit function (α)



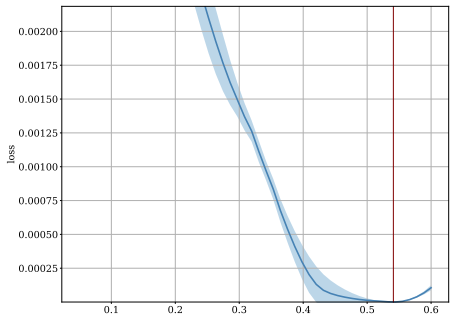
(b) Depreciation rate (δ)



(c) Equity issuance cost (λ)



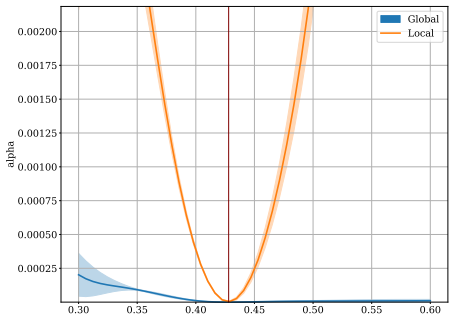
(d) Auto-corr. of productivity (ρ)



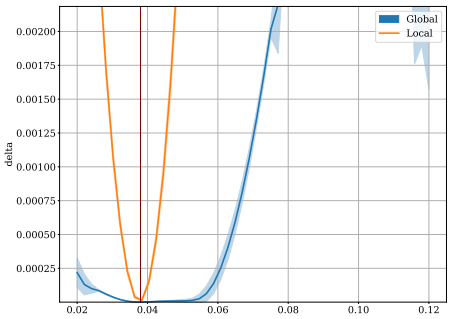
(e) Productivity innovation (σ_z)

This figure shows the minimum loss function for all parameters, with each panel corresponding to a different parameter. In each panel, we use a global optimization routine to minimize the loss function given by Eq. (18), the distance between model and data moments, for all parameters other than the one denoted in the caption. We compute the minimum loss function for the network parameters of the last ten iterations and compute the mean and standard deviation across these ten versions. We show two standard deviation bands along with means. The red vertical line denotes the estimated parameter value. We set the batch size to 512, the replay buffer size to 50,000, the dataset size to 5,000, the learning rate to $1e^{-3}$, and an epoch to 100 iterations. Our neural networks have 2 hidden layers, each with 256 nodes. We use a Sigmoid Linear Unit (SiLU) activation function, defined as $silu(x) = x\sigma(\mathbf{x})$, where σ is the sigmoid function. We run the algorithm for 15 minutes.

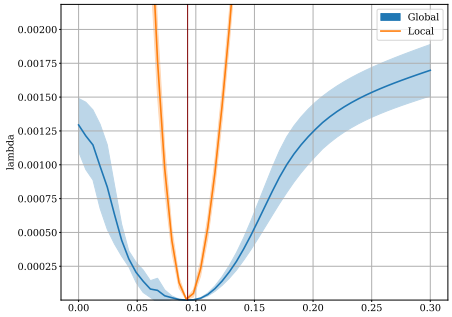
Figure 8: Minimum Loss Function: Local vs Global



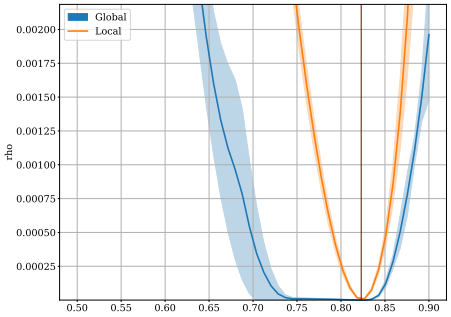
(a) Curvature of profit function (α)



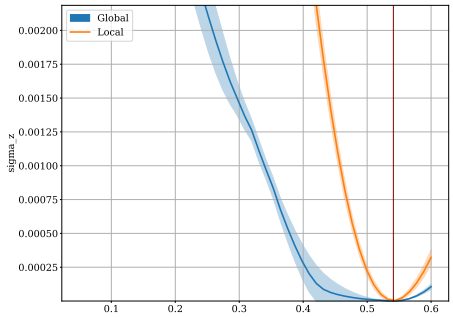
(b) Depreciation rate (δ)



(c) Equity issuance cost (λ)



(d) Auto-corr. of productivity (ρ)



(e) Productivity innovation (σ_z)

This figure shows the minimum loss function for all parameters, with each panel corresponding to a different parameter. In each panel, we use a global optimization routine to minimize the loss function given by Eq. (18), the distance between model and data moments, for all parameters other than the one denoted in the caption. We compute the minimum loss function for the network parameters of the last ten iterations and compute the mean and standard deviation across these ten versions. We show two standard deviation bands along with means. The red vertical line denotes the estimated parameter value. We set the batch size to 512, the replay buffer size to 50,000, the dataset size to 5,000, the learning rate to $1e^{-3}$, and an epoch to 100 iterations. Our neural networks have 2 hidden layers, each with 256 nodes. We use a Sigmoid Linear Unit (SiLU) activation function, defined as $silu(x) = x\sigma(\mathbf{x})$, where σ is the sigmoid function. We run the algorithm for 15 minutes.

Table 1: Moments Estimates

	Data Moments	Model-Implied Moments
Mean(investment/assets)	0.078	0.078
Mean(profit/assets)	0.127	0.126
Mean(equity issuance/assets)	0.042	0.043
Serial corr of profit/assets	0.517	0.521
Std. dev. of profit/assets residuals	0.081	0.081

Notes: This table reports data and model-implied moments. Data moments are based on a sample of non-financial firms from Compustat between 1970 and 2019. Model-implied moments are from a simulation of 16,384 firms over 250 time periods. We solve the model, approximate moments, and estimate parameters using the algorithm described in Section 3. We set the batch size to 512, the replay buffer size to 50,000, the dataset size to 5,000, the learning rate to $1e^{-3}$, and an epoch to 100 iterations. Our neural networks have 2 hidden layers, each with 256 nodes. We use a Sigmoid Linear Unit (SiLU) activation function, defined as $silu(x) = x\sigma(\mathbf{x})$, where σ is the sigmoid function. We run the algorithm for 15 minutes.

Table 2: Parameter Estimates

	α	δ	λ	ρ	σ_z
This paper	0.428	0.038	0.093	0.823	0.541
	(0.0020)	(0.0003)	(0.0015)	(0.0043)	(0.0023)
VFI + differential evolution	0.423	0.038	0.093	0.824	0.539
	(0.0023)	(0.0003)	(0.0022)	(0.0042)	(0.0021)

Notes: This table reports estimated parameters, with standard deviations in parenthesis. Estimates are based on a sample of non-financial firms from Compustat between 1970 and 2019. α is the curvature of the profit function, δ is the depreciation rate, λ is the cost of equity issuance, ρ is the autocorrelation of productivity, and σ_z is the standard deviation of the productivity innovation. The first row shows estimates obtained using the algorithm described in Section 3 and the second row shows estimates obtained by solving the model using value function iteration and estimating parameters with differential evolution. We set the batch size to 512, the replay buffer size to 50,000, the dataset size to 5,000, the learning rate to $1e^{-3}$, and an epoch to 100 iterations. Our neural networks have 2 hidden layers, each with 256 nodes. We use a Sigmoid Linear Unit (SiLU) activation function, defined as $silu(x) = x\sigma(\mathbf{x})$, where σ is the sigmoid function. We run our algorithm for 15 minutes and differential evolution for 24 hours.