Taming the Curse of Dimensionality: Quantitative Economics with Deep Learning[*]

Jesús Fernández-Villaverde
University of Pennsylvania, NBER, CEPR
Galo Nuño
Banco de España, CEPR, CEMFI
Jesse Perla
University of British Columbia

October 29, 2024

Abstract

We argue that deep learning provides a promising avenue for taming the curse of dimensionality in quantitative economics. We begin by exploring the unique challenges posed by solving dynamic equilibrium models, especially the feedback loop between individual agents' decisions and the aggregate consistency conditions required by equilibrium. Following this, we introduce deep neural networks and demonstrate their application by solving the stochastic neoclassical growth model. Next, we compare deep neural networks with traditional solution methods in quantitative economics. We conclude with a survey of neural network applications in quantitative economics and offer reasons for cautious optimism.

JEL classification: C61, C63, E27.

Keywords: Deep learning, quantitative economics.

[*]This paper is based on Jesús Fernández-Villaverde's keynote address at the 2024 Econometric Society Interdisciplinary Frontiers (ESIF) conference on Economics and AI+ML at Cornell. We are very grateful to Douglas de Araujo, Matthias Rottner, Simon Scheidegger, and Yucheng Yang for their comments. The views expressed in this manuscript are those of the authors and do not necessarily represent the views of the Banco de España or the Eurosystem. All remaining errors are ours.

1 Introduction

In this paper, we explore how deep learning tools can help address challenges in quantitative economics that were previously considered intractable. Specifically, deep learning can aid in solving high-dimensional dynamic equilibrium models (DEMs), which underpin fields as diverse as macroeconomics, IO, international economics, game theory, and finance.

Economists are interested in the "solution" of DEMs (Section 2 will clarify what it means to "solve" a model). This process typically involves finding numerical approximations of unknown endogenous functions, such as value and policy functions, conditional expectations, and distributions of agents. However, these functional approximations are hampered by the curse of dimensionality (Bellman, 1957, p. IX), which limits the complexity of the DEMs that can be solved to models with only a few state variables. Deep learning promises that, if properly applied, it can "tame" the curse of dimensionality in many cases.[1]

We aim to make four key points. Our first point is that much of the ongoing revolution in artificial intelligence stems from the use of deep neural networks (NNs), that is, NNs with multiple hidden layers. While the origins of NNs date back to the 1940s and 1950s (Rosenblatt, 1958), the current surge in interest began in 2012, when Krizhevsky et al. (2012) demonstrated that a deep convolutional NN (a specific deep NN architecture we will introduce below) could outperform existing methods in image recognition.[2]

A series of remarkable achievements followed this breakthrough. Mnih et al. (2015) proposed a new reinforcement learning algorithm based on deep NNs that learned to play Atari video games, surpassing human players in some of them. Silver et al. (2016) developed another reinforcement learning algorithm that mastered the game of Go, which had long been considered too complex for artificial intelligence. Vaswani et al. (2017) introduced the transformer architecture, opening the door to the development of large language models (LLMs) such as ChatGPT and text-to-image models such as

[1]We pick the term "tame" and not "break" because the curse of dimensionality always shows up in worst-case scenarios. Our goal is to demonstrate under what conditions the curse can be managed.

[2]In the literature on artificial intelligence, "deep learning" commonly refers to computer programs that use deep NNs and improve performance on a task through experience (Mitchell and Mitchell, 1997, and Murphy, 2022, Section 1.1). Although some of the algorithms we use may involve subtle elements of learning, it is more accurate to describe our work as utilizing tools from the deep learning toolkit, which may or may not involve solving a formal learning problem.

Dall-E. More recently, Trinh et al. (2024) demonstrated how an LLM could be trained to solve mathematical theorems at the level of the Mathematical Olympiad.[3]

Unsurprisingly, given the success of deep learning in other fields, economists have begun exploring what deep NNs can achieve in economics. The answer has been "a lot," ranging from novel empirical methods to innovative approaches for handling DEMs. In this paper, we focus on the latter —specifically, how deep NNs assist economists in solving DEMs used across many fields.

Our second point is that once the unfamiliar jargon of deep learning is stripped away, one realizes that the tools from NN are generalizations of methods already familiar to economists. For instance, we will highlight how linear regression is simply a special case of an NN. Similarly, we will show that NN-based functional approximations bear a strong resemblance to traditional projections, such as those using Chebyshev polynomials. However, there are subtle but important differences between deep learning techniques and conventional methods that can significantly affect how the curse of dimensionality is managed.

Our third point is that, despite these similarities, economists must always remember that we are dealing with DEMs, where we need to impose equilibrium conditions: the restrictions across parameters in agents' decision rules that Hansen and Sargent (1980, p. 37) called "the hallmark of rational expectations models." The feedback between agents' optimizing behavior and the consistency requirements of equilibrium generates challenges that are absent in the typical applications of deep learning in fields such as computer science, natural sciences, engineering, and statistics. While quantitative economists can benefit from deep learning, we must remain mindful that we are computing equilibria (or solving social planner problems) and adapting deep learning tools accordingly.

Our fourth point is that the unreasonable effectiveness of deep NNs in economics is not a product of "magic," but is based on intuitive geometrical reasoning.[4] In particular, NNs geometrically transform the original state space of the model into an efficient representation that captures the central payoff-relevant features of the states. In both economics and other fields, the key to success in functional approximation

[3]Because of these recent successes and space limitations, we will focus nearly exclusively on deep NNs, with only some passing references to other machine-learning (ML) methods. Most of our arguments also hold for the broader class of ML tools. We will highlight this generality often.

[4]Our geometrical interpretation of NNs is deeply influenced by Bronstein et al. (2021). The interested reader can find much more on this topic at `https://geometricdeeplearning.com/`.

often lies in identifying the right space for the approximation, not in searching for better approximating functions.[5]

A trivial example illustrates the point. The Cobb-Douglas production function, $y = k^{\alpha} l^{1-\alpha}$, links output, y to capital, k, and labor, l. One of the first things that we explain to students about this function is that it is much easier to work with it in logs, $\log y = \alpha \log k + (1-\alpha) \log l$. Taking logs is simply moving the problem from one space (the levels of k and l) to a more convenient space ($\log k$ and $\log l$) where the function is linear. NNs generalize this idea to the representation of the state variables of arbitrary models. As Tom Sargent loves to say, "[f]inding the state [of a model] is an art." NN helps with this art, as it allows geometric transformations of complex models where our economic understanding of the mathematical structure of the problem does not reach. There is no magic, just better geometry.

Moreover, efficient representations capture features of the underlying state space, which leads to good generalization (e.g., off-equilibrium policies for the agents that are consistent with economic intuition). If one can ensure sufficient variation in points within the space of representations, the functional approximation may cover an economically significant portion of the original state space.[6]

To illustrate this idea, we will discuss the central role that regularization plays in deep learning methods. Regularization includes all the steps taken by the researcher to avoid overfitting the data. This can be done explicitly (e.g., by adding a penalty term in the objective function that discourages overly complex models) or implicitly (e.g., with the selection of the optimization algorithm that will be used to "train" the deep NN). Thanks to regularization, we will get a good generalization. Without regularization, simply replacing traditional basis functions like orthogonal polynomials or splines with deep NNs does not improve matters much. In short, deep learning succeeds because it provides better geometry, not because it stumbles upon a "clever" basis function for approximation.

[5]Contrary to some common misconceptions, deep learning is not primarily about dimensionality reduction. One classic example of this is Cover's theorem, which motivates the use of kernel methods by showing how embedding low-dimensional data in high-dimensional space makes it easier to find separating hyperplanes for classification (Cover, 1965).

[6]Economists already use a similar idea in the method by Krusell and Smith (1998), which selects one or a few moments of the distribution of heterogeneous agents to track the evolution of the distribution, or in the oblivious equilibrium solution concept, which solves for best responses to the averages of the behavior of other players (Weintraub et al. 2008, Benkard et al. 2015, and Weintraub et al. 2010). Deep learning makes it possible for the representation space to be learned and approximated rather than "engineered" by the economist.

Before diving into the main body of the paper, we offer three important caveats. First, our experience is that economists often find the formal probabilistic perspective to deep learning, such as the one followed by Murphy (2022, 2024), easier to relate to their existing methods than other perspectives emphasizing engineering concepts such as circuits or biological analogies such as neurons. Thus, we will avoid jargon about "neurons," "activation," and "rectifier units" and stick to the more familiar language of probability, linear algebra, and optimization. Classic textbooks, such as Goodfellow et al. (2016), may provide helpful alternative perspectives to ours. For instance, one promising but unexplored perspective here is to view learning through the lens of compression and complexity.

Second, we must emphasize that the interaction between quantitative economics and deep learning is vast. Given the space limitations of this paper, we can only scratch the surface of the key ideas. To keep the discussion concise, we will often omit the detailed technical specifics. We will provide many references, and we hope the interested reader will be sufficiently intrigued by our exposition that she will check these references for details. We are tremendously excited about deep learning in quantitative economics and want to convey that sense of discovery without getting bogged down in the minutiae.

Third, we will focus exclusively on how quantitative economics and deep learning interact, as we do not have the space to cover the relationship between deep learning and econometrics or finance. Some recent surveys about the use of deep learning techniques and other ML tools in economics can be found in Chakraborty and Joseph (2017), Athey and Imbens (2019), Scheidegger and Bilionis (2019), Nagel (2021), Kelly and Xiu (2023), de Araujo et al. (2024), and Dell (2024).

The rest of the paper is organized as follows. Section 2 outlines what the solution of a DEM is and why it is challenging to find it. Section 3 introduces the basic ideas of NNs, how to train them, and explains why they work in practice. Section 4 applies the ideas in the previous sections to a canonical example: the stochastic neoclassical growth model. Section 5 briefly summarizes other applications of NNs in the literature. Section 6 compares NNs with traditional solution methods in economics. Section 7 outlines some reasons for cautious optimism. Section 8 concludes with a glimpse into the future.

2 Solving DEMs

Before delving into what NNs are, we must provide some background on what it means to solve a dynamic equilibrium model (DEM) and where the computational challenges arise. Economists well-versed in DEMs may find much of this section familiar, though some classic ideas might be presented in new ways that offer fresh insights. For computer scientists, our treatment of DEMs could introduce a range of issues that differ from those in other NN applications. In particular, the concept of "equilibrium" in economics is emphatically *not* the same as "equilibrium" in other fields; the same word, totally different meanings.

Models as equilibrium Markov processes (EMPs). In modern DEMs, agents are assumed to solve a Markov decision problem (MDP), maximizing an objective function subject to their budget constraints, technologies, information, and exogenous shocks. This definition is flexible enough to include behavioral agents (e.g., by imposing limits on maximization ability) and most non-Markovian problems (which can be reformulated as an MDP by redefining the state variables). The maximization yields a policy function (or decision rule) that maps the agent's state variables (e.g., the capital and productivity shock) to choices (e.g., consumption).

Next, we take the policy function and combine it with consistency conditions to generate what we call an equilibrium Markov process (EMP). For instance, we use the consumption policy function mentioned earlier to produce sequences of consumption and capital that satisfy the economy's resource constraints. These consistency conditions are typically referred to as the model's "equilibrium conditions."

The term "equilibrium" often confuses those outside of economics, as it is commonly associated with concepts of stability or steady states from the natural sciences. In reality, an EMP can involve highly complex, non-stationary dynamics, markets that do not clear, idle resources, and other phenomena. Moreover, many —if not most— equilibria of interest are not efficient.[7]

An EMP is simply a stochastic process for the endogenous variables of the model that simultaneously satisfies the MDP of the agents and the consistency conditions. In other words, the DEM transforms the exogenous shocks to the economy into an

[7]When economists refer to "partial equilibrium," they typically mean that certain aspects of the EMP, such as the interest rate, remain exogenously fixed.

EMP.[8] Or, more simply, economic theory is a set of constraints on stochastic processes. Finding this EMP is what we call solving the model.

Unfortunately, except for a few special cases with analytic solutions, economists are forced to find the EMP using numerical methods. The MDP is usually nonlinear and stochastic, and when there are strategic interactions among the agents (like in game theory), we need to consider those. Once we have the EMP, we can simulate it to evaluate objects of interest, such as moments of endogenous variables, impulse-response functions, or demand functions, or to compute counterfactuals.

The equilibrium feedback loop. DEMs are subject to an acute curse of dimensionality because we must solve simultaneously for the DMP of the agents and the EMP and explicitly consider the feedback loop between the two (Sargent, 2024). The assumptions that encourage parsimony (Nash equilibria, rational expectations, and related approaches eliminate degrees of freedom in the EMP by tying it to the MDP) are also a central source of their computational difficulty.

In comparison, agent-based models are tractable numerically precisely because they avoid this feedback loop and do not provide notions of self-consistency. However, most economists are suspicious of models without self-consistency because they create arbitrage opportunities. Suppose the evolution of the EMP is payoff-relevant for the agents. In that case, the agents have incentives to forecast with the new EMP when solving for the policies in their MDP and "exploit" the agents that do not do so through trading in the financial markets.

Taming the curse of dimensionality. Given the challenge of having to compute the equilibrium feedback loop of the DEM, quantitative economists have traditionally resorted to brute force: solving for the entire self-consistent solution of functional equations over a large grid of the state space. The exception is perturbation solutions, which only vow accuracy local to a non-stochastic steady state.[9]

NNs promise that, in many contexts (though not all), they can tame the curse of dimensionality in finding EMPs, including the equilibrium feedback loop. If so, this

[8]When the DEM is deterministic instead of stochastic, we have an EMP with a trivial distribution.

[9]Perturbation methods prioritize numerical precision near a deterministic steady state over global applicability. Similarly, adaptive sparse grids efficiently span the state space by focusing on key regions for solving the DEM (Brumm and Scheidegger, 2017). Notably, both perturbation and sparse grids share with deep NNs the strategy of mitigating the curse of dimensionality by concentrating on high-probability regions of the model's state space —a point that will be clarified in Section 4.

could drastically expand the class of DEMs we can handle and the policy-relevant questions we can address. However, explaining why this is the case requires several steps. The first is to present deep NNs formally.

3 Neural networks

In this section, we briefly introduce NNs as function approximators. This overview is not intended to replace a comprehensive textbook exposition. Several excellent texts cover the subject in depth (e.g., Goodfellow et al., 2016, and Murphy, 2022), and we encourage the reader to consult them.

More importantly, the precise definition of NNs and deep learning is less critical here, especially given the various ways they can be described. Instead of getting overly focused on definitions, we suggest the reader focus on two key concepts that will guide our discussion.

First, unlike the traditional approach in economics, which approximates a function $f : \mathcal{X} \to \mathbb{R}$ using a flat functional form, such as a linear combination of basis vectors, NNs approximate functions using nested structures. For instance, we aim to approximate $f(X) \approx g(\phi(X))$, where $\phi : \mathcal{X} \to \mathcal{Z}$, $g : \mathcal{Z} \to \mathbb{R}$, and $\mathcal{Z}$ is the "representation space."

Second, while $\phi(\cdot)$ and $g(\cdot)$ are sometimes found jointly through the same algorithm, we can conceptualize the process as consisting of two distinct stages. The function $\phi(\cdot)$ provides a better geometrical representation of the underlying space $\mathcal{X}$, one that exhibits properties more conducive to downstream tasks like $g(\cdot)$ and generalization. When the space $\mathcal{Z}$ is Euclidean and endowed with meaningful norms, this transformation is called an "embedding," which may reside in a space of lower or higher dimensionality than $\mathcal{X}$. The function $g(\cdot)$ maps the enhanced representation $\phi(\cdot)$ to an output, and it can either be learned or chosen by the researcher.

3.1 Neural networks as function approximators

Imagine we want to approximate ("learn") an unknown function $y = f(X)$, where y is a scalar and $X = \{x_0 = 1, x_1, x_2, ..., x_N\} \in \mathcal{X}$ is a vector, including a constant.[10]

[10]We can accommodate the case where y is a vector using heavier notation. A natural case is when y is a probability distribution.

The x_n's are known as the features of the data and $\mathcal{X}$ is the feature space. The dimensionality N can be potentially large. Returning to the MPD from the previous section, $f(X)$ can be the value function encoding the MPD or its associated policy function, and X is the vector of state variables of the agent. Other functions we may want to approximate in economics can include a pricing kernel, a conditional expectation, and many others.

A basic NN. A single-layer NN is an approximation to $f(X)$ of the form:

$$y = f(X) \approx g^{NN}(X;\theta) = \theta_0 + \sum_{m=1}^{M} \theta_m \phi(z_m) \tag{1}$$

where $\phi(\cdot)$ is an activation function and $z_m = \sum_{n=0}^{N} \theta_{n,m} x_n$. The $\phi(z_m)$'s are called the representations of the data, and M is the width of the model. Also, $\theta = (\theta_0, ..., \theta_M, \theta_{0,1}, ..., \theta_{N,M})$ are the weights of the network (also known as parameters, although one must distinguish them from the structural parameters of the model, which are objects with clear economic interpretation such as risk aversion or the discount factor). Some popular activation functions include the sigmoidal function, $\phi(z) = \frac{1}{1+e^{-z}}$, the hyperbolic tangent, $\phi(z) = \tanh(x)$, the rectified linear unit (ReLU), $\phi(z) = \max(0, z)$, or the softplus, $\phi(z) = \log(1 + e^z)$. When the activation function is linear, $\phi(z) = \alpha_0 + \alpha_1 z$, the NN is just a linear regression.

We emphasize that equation (1) is a chain of geometric transformations: it takes the features of the data, X, builds M affine transformations of them, $z_m = \sum_{n=0}^{N} \theta_{n,m} x_n$, deforms the shape of these affine transformations using an activation function, $\phi(\cdot)$, to obtain M representations, and recombines these representations into another affine transformation $g^{NN}(\cdot)$. In such a way, we combine two linear combinations with one (potentially) nonlinear operation. The notation $g^{NN}(X;\theta)$ highlights that the resulting approximation depends on both X and θ. Section 3.2 below will explain how to determine these weights. We call this a single-layer NN because there is only one level of transformation from X into y.

NNs were named because their structure resembles the biological neurons in animal brains. While biological analogies have informed some developments in ML, as in the case of convolutional NNs (CNNs), we do not find much value in pursuing the biological analogy, and we find it more of a source of confusion than enlightenment.

A comparison with projection. We can contrast our functional approximation with a shallow NN in equation (1):

$$f(X) \approx g^{NN}(X;\theta) = \theta_0 + \sum_{m=1}^{M} \theta_m \phi\left(\sum_{n=0}^{N} \theta_{n,m} x_n\right), \tag{2}$$

with a functional approximation with a standard projection:

$$f(X) \approx g^{CP}(X;\theta) = \theta_0 + \sum_{m=1}^{M} \theta_m \phi_m(X), \tag{3}$$

where ϕ_m is, for example, a Chebyshev polynomial.

Both functional approximations are surprisingly similar: as we mentioned in the introduction, once we drop the jargon of deep learning, one realizes that NNs are generalizations of methods familiar to economists. Equation (2) has only one function $\phi(\cdot)$, while we have a set of Chebyshev polynomials in equation (3). In exchange, equation (2) requires the additional parameters $\theta_{n,m}$, which are not needed in equation (3). How we determine in practice the θs in each approximation will also be different (e.g., minimizing the loss function (4) vs. a collocation). We will return to this point when we talk below about the training of the network and in Section 6, when we discuss some distinctive features of ML tools.

A deep NN. A deep learning NN is a composition of $J > 1$ NNs (or, if one prefers, of $J > 1$ layers):

$$z_m^0 = \theta_{0,m}^0 + \sum_{n=1}^{N} \theta_{n,m}^0 x_n$$

$$g^{NN(1)}\left(X;\theta^1\right) = z_m^1 = \theta_{0,m}^1 + \sum_{m=1}^{M^{(1)}} \theta_m^1 \phi^1\left(z_m^0\right)$$

$$\dots$$

$$\begin{aligned} y \approx g^{DL}(X;\theta) &= \theta_0^J + \sum_{m=1}^{M^{(J)}} \theta_m^J \phi^J\left(z_m^{J-1}\right) \\ &= g^{NN(J)}\left(g^{NN(J-1)}\left(\dots g^{NN(1)}\left(X;\theta^1\right)\dots;\theta^{(J-1)}\right);\theta^J\right), \end{aligned}$$

where now the weight vector $\theta = (\theta^1, .., \theta^J)$ includes the weights in each layer. The widths $M^{(1)}, M^{(2)}, ..., M^{(J)}$ and activation functions $\phi^1(\cdot), \phi^2(\cdot), ..., \phi^J(\cdot)$ are possibly different across each layer of the NN. The hyperparameter J determines the depth of the NN. If $J = 1$, we have a standard "shallow" NN, whereas $J > 1$ is a deep neural network. Here, "deep" means extending far from the start of the NN at the first layer until the Jth layer, not being profound or penetrating (as in an elaborate philosophical statement). Notice the superindex DL in our notation $g^{DL}(X; \theta)$. Also, from now on, we will focus on deep NNs, although all the arguments we will present also hold for shallow NNs.

Deep NNs are sometimes known as deep feedforward NNs or fully connected, multilayer perceptrons. The "feedforward" comes from the fact that the composition of NNs can be represented as a directed acyclic graph. Below, we will discuss alternative NN architectures.

Like shallow NNs, deep NNs are chained geometric transformations, where we repeatedly build affine transformations and deform them through activation functions. This approach (as we will formalize in the next section) can capture even the most intricate nonlinear shapes while maintaining a linear approximation structure, aside from the activation function —keeping the computations efficient.

Other architectures. There is a rich set of NN architectures beyond the densely connected deep NN we have presented. These include, among others, CNNs, generalized adversarial networks (GANs), and transformers. Later we will argue that there is a fundamental connection among all these different architectures. Suffice it to say at this moment that the choice of architecture shall be dictated by the economics of the problem at hand, in particular, the sorts of representations they might be good at capturing.

CNNs are typically employed to deal with image classification. Regular deep NNs do not work well with a grid-like topology (e.g., 2-D images) because the method is not invariant to shifts. Imagine that we have two pictures of the same object, but in one of them, the object is shifted. A densely connected deep NN often fails to recognize both pictures as equal. CNNs avoid this problem by considering a particular structure on hidden layers that makes them translation-invariant. In economics, CNNs have been used to handle images (Voth and Yanagizawa-Drott, 2024) or optical character recognition in historical documents (Shen et al., 2021).

GANs are employed to generate new data —typically images but potentially any other form of data— with the same characteristics as a training set of original data (Goodfellow et al., 2014). The idea is to have two NNs: a generator network that generates candidates (e.g., images) and a discriminative network that evaluates whether the candidates are real or generated. Both NNs play a minimax game against each other. In economics, GANs have been used, for instance, for estimation (Kaji et al., 2023).

A transformer is a deep learning architecture based on an attention mechanism (Vaswani et al., 2017). Transformers are the basis of the LLMs such as ChatGPT and BERT. The input text is split into n-grams encoded as "tokens." At each layer, each token is then contextualized within the scope of a "context window" via an attention mechanism, allowing the signal for key tokens to be amplified and the signal for less important tokens to be diminished. The applicability of these LLMs extends beyond text, and extensions have been implemented to deal with complex topics such as protein folding (Jumper et al., 2021) or theorem-proving (Trinh et al., 2024).

More general high-dimensional approximations. An NN is a particular example of a more general class of high-dimensional approximations. As such, there is nothing special about it in comparison with many other techniques in ML. However, the concreteness of NNs helps our exposition. We will return to this point in Section 6.

3.2 Training the NN

Training the NN means to select θ such that $g^{DL}(X;\theta)$ is as close to $f(X)$ as possible given some relevant metric (e.g., the ℓ_2 norm). Later, we will come back to explore how this can be done for EMP, since neither y nor X is observable, but instead, they are endogenous equilibrium objects in the model, such as a value or policy function, or state variables. However, the reader shall already notice that applying deep NN to solve DEMs is inherently different than using them to estimate a function from observations, as is done in econometrics.

Loss function. The first step for training an NN is to specify a loss function that sets up a distance between $f(X)$ and $g^{DL}(X;\theta)$ given data on outcomes $\mathbf{Y} = \{y_1, y_2, ..., y_L\}$ of the function and features $X = \{X_1, X_2, \ldots, X_L\}$: $y_l = f(X_l)$ for $l = 1, \ldots, L$.

The data are often called the training sample. In most deep learning applications in the natural sciences, engineering, and econometrics, the data come from the real world. In Section 4, the data will be the results of simulating our DEM or selecting a grid over the state or sequence space. This means that, for DEM models, the only limitation to the sample size L will be the storage capacity and the computing time, providing a welcomed degree of flexibility.

A natural loss function is the quadratic loss function $\mathcal{E}\left(\theta;\mathbf{Y},\widehat{\mathbf{y}}\right)$ on the differences between the data on outcomes and the forecast of the NN:

$$\begin{aligned}\theta^* &= \arg\min_\theta \mathcal{E}\left(\theta;\mathbf{Y},\widehat{\mathbf{y}}\right) \\ &= \arg\min_\theta \sum_{l=1}^{L}\mathcal{E}\left(\theta;y_l,\widehat{y}_l\right) \\ &= \arg\min_\theta \frac{1}{2}\sum_{l=1}^{L}\left\|y_l-g^{DL}\left(X_l;\theta\right)\right\|^2, \qquad (4)\end{aligned}$$

where $\widehat{y}_l = g^{DL}\left(X_l;\theta\right)$ and $\widehat{\mathbf{y}}=\{\widehat{y}_1,\widehat{y}_2,\ldots,\widehat{y}_L\}$.

However, other loss functions can be used. For instance, we can add regularization terms of the form $\lambda\sum_{i=1}|\theta_i|$ (LASSO), $\lambda\sum_{i=1}\theta_i^2$ (ridge regression), or a combination $\lambda_1\sum_{i=1}|\theta_i|+\lambda_2\sum_{i=1}\theta_i^2$ (elastic nets). As we mentioned in the introduction, regularization plays a key role in the success of deep NNs, although it is often achieved through the optimization algorithm, not through an explicit term.[11] Thus, let us move to exploring how to minimize the loss function.

Minimizing the loss function. The minimization in equation (4) is usually accomplished with (first-order) descent direction iteration methods. Starting at a point in the weights space $\theta^{(1)}$, a descent direction algorithm generates a sequence of steps (called iterates) that converge to a local minimum as follows:

1. At iteration k, check whether $\theta^{(k)}$ satisfies a termination condition. If so, stop; otherwise, go to step 2.

2. Determine the descent direction $\mathbf{d}^{(k)}$ using local information such as gradient or Hessian.

[11] See Smith et al. (2021), Chiang et al. (2022), and Zhang et al. (2021).

3. Compute step size $\alpha^{(k)}$.

4. Compute the next candidate point: $\theta^{(k+1)} \leftarrow \theta^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}$.

The choice of the step size α and the direction $\mathbf{d}$ determines the "flavor" of the different algorithms. We analyze each of them in turn.

Step size. Given $\mathbf{d}$, one way to set the step size is to solve a line search:

$$\alpha^k = \arg\min_{\alpha} \mathcal{E}(\theta^{(k)} + \alpha\mathbf{d}^{(k)})$$

with any standard method. Under this choice, it can be shown that $\mathbf{d}^{(k+1)}$ and $\mathbf{d}^{(k)}$ are orthogonal. In practice, however, line search can be costly, and we can settle for a fix α, an α^k that geometrically decays, or an approximated line search.

Direction. A natural choice for $\mathbf{d}$ is the direction of the steepest descent given by the direction opposite the gradient, $\nabla\mathcal{E}(\theta) = \sum_{l=1}^{L} \nabla\mathcal{E}\left(\theta; y_l, \widehat{y}_l\right)$, computed over all the training samples $\nabla\mathcal{E}\left(\theta; y_l, \widehat{y}_l\right) \equiv \left[\frac{\partial\mathcal{E}}{\partial\theta^0_{0,0}}, \frac{\partial\mathcal{E}}{\partial\theta^0_{0,1}}, ..., \frac{\partial\mathcal{E}}{\partial\theta^J_{N,M}}\right]^\top$, and normalized $\mathbf{d}^{(k)} = -\frac{\nabla\mathcal{E}(\theta^{(k)})}{||\nabla\mathcal{E}(\theta^{(k)}).||}$ using an appropiate norm $||\cdot||$. If the loss function is smooth and the step size is small, the method leads to improvement as long as the gradient is not zero.

The initial θ is typically determined by domain knowledge (i.e., the researcher's understanding of what can be sensible initial weights of the NN) or drawn from some probability distribution (such as a Gaussian distribution).

Stochastic gradient descent and minibatch. Evaluating the gradient for the whole training set can be very costly, as we may be dealing with L in the thousands or even millions. This is why neural networks are typically trained using variants of the stochastic gradient descent (SGD) algorithm (Robbins and Monro, 1951).

SGD uses only one data point l to evaluate an approximation to the gradient $\nabla\mathcal{E}(\theta) = \nabla\mathcal{E}\left(\theta; y_l, \widehat{y}_l\right)$. The data point is randomly selected in each iteration and, thus, it varies in each step. SGD trades off a slower convergence rate for faster computation. It converges almost surely to a global minimum when the objective function is convex (and to a local minimum otherwise). It converges exponentially fast to a neighborhood of the solution and, then, bounces around a "zone of confusion." An additional advantage is that, by introducing a noisy update process, SGD can avoid getting stuck in local minima.

In practice, it is usually better to employ a minibatch gradient descent. A minibatch is a compromise between using the whole training set and pure SGD by considering a batch of samples with a size $L \gg B > 1$:

$$\nabla\mathcal{E}(\theta) = \sum_{l=1}^{B} \nabla\mathcal{E}\left(\theta; y_l, \widehat{y}_l\right).$$

Momentum. Gradient descent can perform poorly in "narrow valleys" of the loss function, as it may require many steps to make progress. Unfortunately, these narrow valleys are not exotic occurrences, since we often minimize over hundreds of thousands of weights. If the function to minimize has flat areas, one can introduce a momentum update equation:

$$\begin{aligned} v^{(k+1)} &= \beta v^{(k)} - \alpha g^{(k)} \\ \theta^{(k+1)} &= \theta^{(k)} + v^{(k+1)}. \end{aligned}$$

for some α and β. This modification reverts to the gradient descent version if $\beta = 0$. Intuitively, the momentum update is like a ball rolling down an almost horizontal surface. Momentum prevents the ball from getting stuck in a local valley.[12]

A very successful optimization algorithm based on momentum, and perhaps the most popular of all optimization methods in deep learning, is `Adam` (Adaptive Moment Estimation) by Kingma and Ba (2017). `Adam` uses running averages of both the gradients and the second moments of the gradients:

$$\begin{aligned} m^{(k+1)} &= \gamma_1 m^{(k)} + (1-\gamma_1)\nabla\mathcal{E}(\theta^{(k)}) \\ v^{(k+1)} &= \gamma_2 v^{(k)} + (1-\gamma_2)\left(\nabla\mathcal{E}(\theta^{(k)})\right)^2 \\ \theta^{(k+1)} &= \theta^{(k)} - \eta\frac{\widehat{m}}{\widehat{v}+\epsilon} \end{aligned}$$

where $\widehat{m} = \frac{m^{(k+1)}}{1-\gamma_1}$ and $\widehat{v} = \sqrt{\frac{v^{(k+1)}}{1-\gamma_2}}$.

[12]A quick intro to momentum can be found at `https://fa.bianp.net/blog/2021/hitchhiker/` and a more subtle interpretation at `https://distill.pub/2017/momentum/`.

Automatic differentiation. We can easily evaluate the loss function $\mathcal{E}(\theta^*; Y, \widehat{\mathbf{y}})$ and the gradient $\nabla\mathcal{E}(\theta^*; Y, \widehat{\mathbf{y}})$ for a given θ^* and an arbitrary deep NN using automatic differentiation. For example, for the gradient, we can use backpropagation, which is just letting the computer execute the chain rule (Rumelhart et al., 1986). If a function is $\mathcal{E} : \mathbb{R}^N \to \mathbb{R}$ for large N, the chain rule can be executed backward to find gradients with a computational complexity independent of N using what is called "reverse-mode auto-differentiation" (sometimes referred to as "backprop").

The tradeoff is that while the computational cost of calculating gradients is largely independent of the size of θ, the code must store all of the intermediate values when calculating the loss function, which leads to significant memory requirements for large θ or large data. This is a key reason why using SGD or minibatch is so important, in particular if one is training the deep NN on a GPU, where memory is precious.

3.3 Designing the NN architecture

So far, we have taken many aspects of the NN architecture as given. But in practice, one also needs to design them. These parameters are commonly known as the NN hyperparameters, and they include the activation function $\phi(\cdot)$, the number of neurons M, the number of layers J, and the size of the minibatches B.

There are several complementary approaches to selecting hyperparameters. For example, one can experiment with different combinations of M, J, and B until the error function $\mathcal{E}(\theta; \mathbf{Y}, \widehat{\mathbf{y}})$ is as small as possible. Cross-validation is a more formal approach for the same task: we split the training sample into multiple parts, training the NN in some of those and employing the remaining parts of the training sample to check the NN's forecast ability. Another possibility is to drop some of the connections in the NN randomly (i.e., making some of the weights in θ equal to zero) and check that the performance of the NN does not degrade. An absence of degradation indicates that the architecture of the NN is sufficiently rich to be robust to small deviations.

Gauging the quality of the architecture is not different from the testing we need to undertake with any other numerical approximation. For instance, when we implement a value function iteration, we must decide how many points to include in the grid of the state variables and how to space them. Similarly, when we approximate a function with Chebyshev polynomials, we need to determine the order of the approximation and how to weigh the residuals of the approximation to compute the unknown coefficients of such an approximation. That is, tuning hyperparameters for NN is no more or less challenging than in other areas of computational economics.

3.4 Why do NNs work?

NNs are not "black boxes." Instead, NNs move the original functional approximation problem, which usually lies in a difficult-to-characterize space, to an equivalent approximation problem that lives in a much more convenient space. Before presenting this argument more carefully, we need to introduce two results that ensure that, at least in principle, NNs can deliver the results we search for: NNs are universal approximators that break the curse of dimensionality.

The universal approximation theorem. NNs are universal approximators: they can approximate any Borel measurable function mapping finite-dimensional spaces arbitrarily well (Hornik et al. 1989, and Cybenko 1989). In other words, NN can approximate any standard function used in economics, including those with jumps or non-differentiabilities. Unfortunately, this result is of little practical use, as it does not provide guidance on how to implement NNs in real-life applications.

Breaking the curse of dimensionality. While it is reassuring that NNs are universal approximators, there are other well-known approximators, such as Taylor, Fourier, or Chebyshev series, that can also approximate large classes of functions. Thus, what is special about NNs? A possible answer to that question comes from a number of theorems that show how NN can break the curse of dimensionality (Barron, 1993, and Bach, 2017). In particular, a shallow NN of width M achieves integrated square errors of order $\mathcal{O}(1/M)$. In comparison, for series approximations, such as a Chebyshev projection, the integrated square error is of order $\mathcal{O}(1/(M^{2/N}))$ where N is the dimensions of the function to be approximated. The key difference is that the dimensionality of the inputs does not affect the accuracy of the network. Equivalent theorems hold for deep NNs, usually with tighter upper bounds.

As before, this result is of limited practical use. To properly train an NN, one needs good representations of the data (i.e., the transformation $\phi(x)$) and regularization. As dimensionality increases, it becomes impossible to have high accuracy over the entire space of possible data or states and one can always design worst-case scenarios where it is not possible to generate these good representations and regularization. Contrary to often-heard statements, this problem is not about the quantity of data available. For example, one can fit an NN with 50k parameters with two data points in 512 dimensions (Ebrahimi Kahou et al., 2021). In fact, Nakkiran et al. (2019) show that, in many contexts, having more data can hurt.

Deep learning and geometry. The best argument to account for the success of deep NNs is that they search for convenient geometrical representations of high-dimensional manifolds by applying chained geometric transformations to the features of the data. In other words, we search for a good representation space.

We borrow a well-known example from Chollet et al. (2022, Sec. 2.3.6). Imagine that you have crumpled a sheet of paper into a ball. The paper ball is the function you want to approximate in the computer, but doing so, with all the ball's irregularities, is hard. That is why standard methods like splines or Chebyshev polynomials fail as soon as we are dealing with an irregular ball in four or five dimensions. The geometric transformations of the NN expand the paper ball (the affine transformation) and unfold it (the activation function) until it is a plain sheet of paper again. Approximating a sheet of paper (a linear plane) in a computer is straightforward.

This geometric interpretation connects with the so-called "manifold hypothesis," which posits that many high-dimensional data from real-world applications actually lie along low-dimensional manifolds inside the high-dimensional space where they are encoded (Cayton, 2005). Under this hypothesis, deep learning only has to fit those low-dimensional manifolds within their potential input space. A related line of research looks for even deeper links between geometry and deep learning by trying to link the different neural network architectures to different symmetries (Bronstein et al., 2021).[13] That is, all NNs share a common geometrical structure, regardless of the details of their architectures. These are fascinating lines of research that may bear fruitful new results in the coming years.

Practical advantages and limitations. NNs present other advantages. First, NNs allow for easy parallelization, an incredible strength given the current ubiquity of graphic processing units (GPUs) (Fernández-Villaverde and Valencia, 2018). Second, there is more and more cutting-edge hardware, such as field programmable gate arrays

[13] Vershynin (2018) and Wainwright (2019) introduce some basic concepts of high-dimensional geometry applied to data. An important point to remember is that intuition from three-dimensional Euclidean spaces is a poor guide for the geometry of high-dimensional spaces.

(FPGAs), that improves the performance of deep learning algorithms by two orders of magnitude (Cheela et al., 2022). Deep NNs and the related calculations of the gradients boil down to matrix-vector products. GPUs and FPGAs are optimized to store matrices and vectors and multiply them in parallel. Finally, deep NNs are easy to code, and there are outstanding open source libraries such as `Tensorflow`, `Pytorch`, or `JAX` that integrate well with easy scripting languages such as `Python`.

Notwithstanding, while NNs can work extremely well, there are no silver bullets. There are clear and serious tradeoffs in real-life applications. Training an NN often requires an understanding of the economic problem at hand. To make this point clearer, next, we will compare traditional functional approximation methods with NN and other ML tools.

4 An example: The stochastic growth model

Although the goal of using deep NNs is to extend the dimensionality of the DEMs we can solve, it is helpful to demonstrate how they function in a low-dimensional example. In such cases, the challenges and benchmark solutions are more transparent. While we would not expect NN methods to outperform traditional approaches in solving low-dimensional DEMs, we will see that the solution still takes just 1.2 seconds on a standard laptop using only one CPU.

The stochastic growth model. We choose as our application the stochastic neoclassical growth model, the backbone of modern macroeconomics. A representative household picks a sequence of consumption c_t and capital k to solve:

$$\max \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \log(c)$$

given a budget constraint $c_t + k_{t+1} = w_t + (1 + r_t - \delta) k_t$, $\forall\ t > 0$, where w_t is the wage paid for the (inelastically supplied) unit of labor of the household and r_t is the rental of capital and a transversality condition:

$$\lim_{T\to\infty} \mathbb{E}\left[\beta^T k_{T+1} c_T^{-1} \;\middle|\; k_0, z_0\right] = 0. \tag{5}$$

A representative firm produces output using capital and the unit of labor supplied by the household with the technology $y_t = z_t^{1-\alpha} k_t^{\alpha}$, subject to productivity shocks $\log z_{t+1} = \rho \log z_t + \sigma \nu_t$, $\nu_t \sim \mathcal{N}(0, \sigma)$. Since input markets are competitive, input prices are equal to marginal productivities. Also, the aggretate resource constraint holds, $y_t = c_t + k_{t+1} - (1-\delta) k_t$.

Let us adopt a recursive notation on the states of the model $X \equiv \begin{bmatrix} k & z \end{bmatrix}^{\top} \in \mathbb{R}_{+}^{2}$. Then, we can derive the Euler equation that characterizes the household problem as a functional equation on the capital accumulation policy function $k'(k, z)$:

$$1 = \mathbb{E}\left[\beta \frac{c(k,z)}{c(k'(k,z), z')} \left(1 - \delta + \alpha z'^{1-\alpha} k'(k,z)^{\alpha-1}\right)\right] \tag{6}$$

where $c(k, z) \equiv z^{1-\alpha} k^{\alpha} + (1-\delta)k - k'(k, z)$ and the expectation is taken over z'.

Thus, conditioning on a policy, the evolution of the Markov process for X is:

$$X' = \Phi\left(X, \nu; k'\right) = \begin{bmatrix} k'(k,z) \\ \exp(\log z + \sigma \nu) \end{bmatrix}. \tag{7}$$

We can relate our previous equations to Section 2. Equation (6) provides the key conditions on the MDP. The EMP can be found as the stochastic process (7) conditioned on the $k'(\cdot)$ policy that fulfills equation (6). Formally, the EMP is a measure over the trajectories, $\{X_t\}_{t=0}^{\infty} \sim \mathbb{P}_{X_0}$, generated from the $X_{t+1} = \Phi(X_t, \nu_t; k'(\cdot))$ using the equilibrium $k'(\cdot)$, the distribution of ν_t, and the initial condition X_0.

Approximating the policy function. To solve for the EMP, we want to approximate the policy function $k'(k, z)$ with a neural network $k'_{\theta}(k, z)$ that belongs to the architecture $\mathcal{H}(\theta)$ and where θ are the weights of the network. To save on notation, we will write k'_{θ} and let the dependence on (k, z) be left implicit.

The first step is to define a quadratic loss function as the analog of our equation (4). A natural loss function is the difference between the left-hand side and the right-hand side of the Euler equation (6) when, instead of the exact function $k'(k, z)$, we use the approximation k'_{θ}:

$$\mathcal{E}(k'_{\theta}, X) = \left[1 - \sum_{i=1}^{M} \omega_i \left[\beta \frac{c(k,z)}{c(k'_{\theta}, z'(z, \nu_i))} \left(1 - \delta + \alpha (z'(z, \nu_i))^{1-\alpha} (k'_{\theta})^{\alpha-1}\right)\right]\right]^2$$

where $z'(z, \nu_i) = \exp(\rho \log z + \sigma \nu_i)$ is a discretization of the productivity shock using

M Gaussian quadrature notes on $\nu \sim \mathcal{N}(0,1)$ and weights $\{\omega_i\}_{i=1}^M$.

Then, we minimize the expected loss function given the population distribution $\mu^*(X_0, T_{\text{pop}}; k'^*)$ indexed by some initial condition X_0 and a length of the simulation T_{pop}:

$$k'^* = \arg\min_{k'_\theta \in \mathcal{H}(\theta)} \mathbb{E}_{X \sim \mu^*(X_0, T_{\text{pop}}; k'^*)} \left[\mathcal{E}(k'_\theta, X) \right]. \tag{8}$$

The equilibrium feedback loop revisited. Section 2 emphasized that the equilibrium feedback between the MDP of the agents (in our case, the representative household) and the equilibrium constraints made deep learning problems in economics different from applications in other fields. Now is the moment to revisit this argument and to use it to frame our next steps.

In our model, the equilibrium feedback loop appears in equation (8). The policy k'^* shows up on the left-hand side of equation (8) as the solution of the $\arg\min$ and on the right-hand side of equation (8), as determining the distribution with respect to which we take the expectation, $\mu^*(\cdot)$, an equilibrium object that depends on k'^*.

If we knew the true $\mu^*(\cdot)$, then a trivial algorithm to minimize equation (8) would be to sample $\mathcal{D}$ points from $\mu^*(\cdot)$ and solve the "empirical risk minimization," or ERM, i.e., the finite sample version of equation (8):

$$\theta^* = \arg\min_{\theta \in \theta} \frac{1}{|\mathcal{D}|} \sum_{X \in \mathcal{D}} \mathcal{E}(k'_\theta, X).$$

We could also check the generalization properties of k'^* by drawing a different set, $\mathcal{D}_{\text{test}}$ of points from $\mu^*(\cdot)$ and asserting that $\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{X \in \mathcal{D}_{\text{test}}} \mathcal{E}(k'_{\theta^*}, X)$ is small enough.

The challenge, of course, is that we do not know $\mu^*(\cdot)$ precisely because of the equilibrium feedback loop.

Sampling from population distribution. A canonical approach to solve for the equilibrium feedback loop between the distribution and the policies in DEM models at the core of this paper is to iteratively solve an ERM version of equation (8) with samples from a misspecified population distribution and periodically regenerate those samples as the policy function is refined. That is, given a k'_i, we solve:

$$k'_{i+1} \equiv \arg\min_{k'_\theta \in \mathcal{H}(\theta)} \mathbb{E}_{X \sim \mu^*(X_0, T_{\text{pop}}; k'_i)} \left[\mathcal{E}(k'_\theta, X) \right],$$

and then repeat the process with the new k'_{i+1}. Notice, in particular, that we are taking the expectation with respect to the misspecified population $\mu^*(X_0, T_{\text{pop}}; k'_i)$, not the true $\mu^*(X_0, T_{\text{pop}}; k'^*)$.

In our case, we will start with a $k'_1(\cdot)$ policy from the Solow model with constant savings rate s_{ss}, $k'_1(k, z) = s_{ss} z^{1-\alpha} k^\alpha + (1-\delta)k$, and generate $\mathcal{D}_1$ from it, but many other initial guesses are possible.

More formally, our algorithm is as follows:

1. Solve the empirical risk minimization problem given the current $\mathcal{D}_i$,

$$\theta_{i+1} = \arg\min_{\theta \in \theta} \frac{1}{|\mathcal{D}_i|} \sum_{X \in \mathcal{D}_i} \mathcal{E}(k'_\theta, X).$$

2. Generate a new $\mathcal{D}_{i+1}$ using samples from $\mu^*(X_0, T_{\text{pop}}, k'_{\theta_{i+1}})$.

3. Iterate until the empirical risk is below a threshold.[14]

While this algorithm often works, it has several limitations. First, there is no guarantee that the iteration will converge to the true $\mu^*(\cdot)$ and, consequently, to the true policy. The existing convergence results from deep learning on ERM do not apply here due to the endogeneity of $\mu^*(\cdot)$ with respect to the solution of the ERM. Second, and relatedly, the algorithm can be sensitive to the choice of the initial policy, k'_1, used to generate $\mathcal{D}_1$. Specifically, we must ensure sufficient variation across the state space to allow the iterations $k'_{\theta_{i+1}}$ to generalize effectively. Third, even if k'_θ is well approximated, the algorithm might still fail to capture accurately the ergodic distribution. Fourth, it is important to avoid oversampling regions of the state space that imply very small Euler equation errors (e.g., where consumption barely changes across periods) but are associated with near-zero probability in $\mu^*(\cdot)$. These points would reduce the ERM but would not improve generalization.

Parameters, hyperparameters, and optimizer. Before we implement the algorithm above, we need to complete a few preliminaries. First, we need to pick values

[14]We can establish stopping criteria for the optimizer on what is called a "validation set" $\mathcal{D}_{\text{val}}$ sampled using the same process as $\mathcal{D}$, but not directly used in training or $\mathcal{D}_{\text{test}}$. In our case, we stop iterating when the empirical loss, $\frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{X \in \mathcal{D}_{\text{val}}} \mathcal{E}(k'_\theta, X) < 2.5e-5$, which is close to numerical precision. Deep NN algorithms often continue to improve generalization behavior even after the empirical risk is close to zero, so one may not want to stop the optimizer prematurely.

for the parameters of the model. We select $\beta = 0.9$, $\alpha = 1/3$, $\delta = 0.2$, $\rho = 0.9$, and $\sigma = 0.025$.

Next, we select the hyperparameters. First, we use $M = 7$ nodes for quadrature. Second, we design a deep NN with four layers, each with 128 nodes, a ReLU activation function $\max\{0, x\}$, and a softplus final layer, $\log(1 + e^x)$. This leads to approximately $50K$ parameters in θ. Third, we pick a short $T_{\text{pop}} = 20$ and generate trajectories (k_t, z_t) for $t = 0, \ldots, T_{\text{pop}}$ to show that our method can get "right" the transitional dynamics of the model from an initial condition for capital far away from the deterministic steady state. A large T_{pop} might imply that we evaluate a small loss function because we spend many periods around the ergodic distribution (where the Euler equation errors will be small, as the consumption changes little from period to period), even if we have large errors during the transition.

To explore how robust the solution is to variations in the initial conditions, we set: $X_0 \sim \mathcal{N}(\begin{bmatrix} k_0 & z_0 \end{bmatrix}^\top, \Sigma)$ with $z_0 = 1, k_0 = 0.8 \times k_{ss}$ (i.e., we start 20% below the deterministic steady state of capital to induce non-trivial transitional dynamics), and:

$$\Sigma \equiv \begin{bmatrix} 0.008^2 & 0 \\ 0 & 0.02^2 \end{bmatrix}.$$

Finally, in terms of optimization, we will run for 100 iterations of the L-BFGS optimizer and regenerate the $\mathcal{D}_i$ every five iterations. Given the relatively low number of points, there is no need to use more sophisticated stochastic optimizers like SGD or *Adam*. Also, we use $\sigma_{\text{train}} = 0.035 > \sigma = 0.025$, which will speed up convergence by increasing the exploration of the state space, a trick that is less crucial in our application but essential in high-dimensional ones.[15]

The algorithm runs in approximately 1.2 seconds using only the CPU.[16] Since the algorithm is run without any pretraining for the k'_θ with approximately $50K$ parameters, it will sometimes diverge and is automatically restarted.[17]

For comparison purposes, we also compute an approximation of the policy, $k'_{\text{ref}}(k, z)$, with a policy function iteration on the Euler equation along a grid of 50 points in k and 50 points in z.

[15]In harder applications, we can have a schedule to change with iteration number, $\sigma_{\text{train}}(i) \to \sigma$.

[16]In this implementation, we used `JAX` with the `Flax NNX` neural network library and the `optax` optimizer library and ran the code on a standard MacBook Air laptop.

[17]In the results below, the code converged on the third attempt. In many cases, the algorithm will converge in the first attempt and nearly always succeeds by the fifth.

Results. The left panel of Figure 1 shows the initial ten trajectories for $t = 0, \dots T_{\text{pop}}$ used to minimize the loss function and generated from a constant savings Solow policy function. The right panel shows the final 50 trajectories of the test set generated using the final policy function, k'_θ.[18] Clearly, Figure 1 shows that the algorithm has converged to a solution, but how good is this solution?

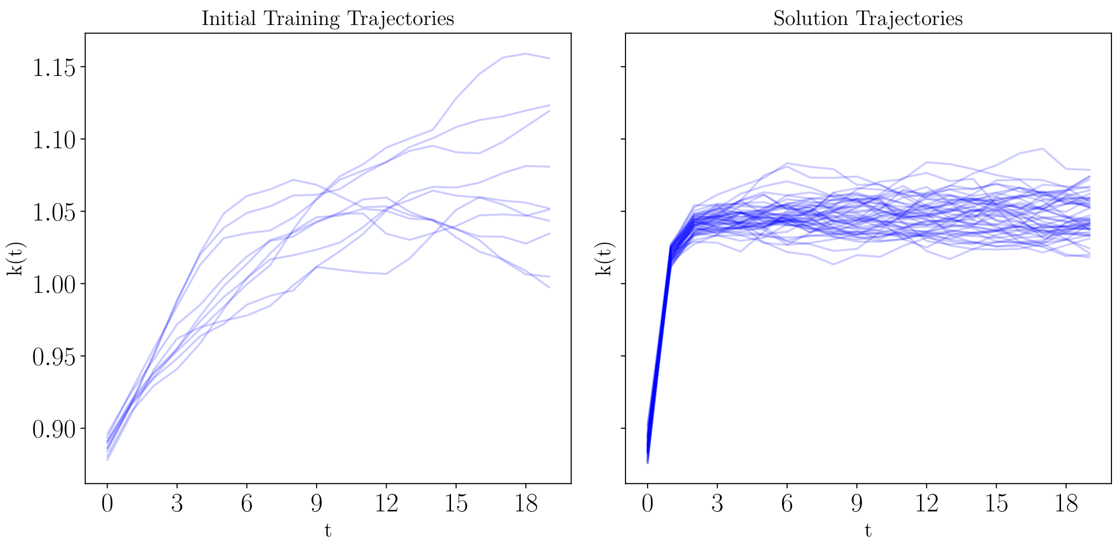

Figure 1: Initial vs. final trajectories of capital.

The answer is: pretty good. The left panel of Figure 2 shows the absolute relative policy error for the initial ten trajectories for $t = 0, \dots T_{\text{pop}}$ used to minimize the loss function. The right panel shows the absolute relative policy error for the 50 trajectories of the test set generated using the final policy function, k'_θ. The policy errors of the solution are low and do not have a significant bias toward small or large t. Furthermore, the low policy errors are not due to the initial condition, which starts with a relative error of up to 15%. Crucial to this success was that the initial dataset explored enough regions of the state space for the algorithm to find the true policy.

To verify this point, we can use the proposed $k'_\theta(\cdot)$ to check an empirical version of the expected loss (8). To do this, we sample from $\mu^*(X_0, T_{\text{pop}}; k'_\theta)$ by drawing X_0 and iterating with equation (7) to form a set of points $\mathcal{D}_{\text{test}}$. We use 50 trajectories to generate a total of 1000 points. Given these samples, the empirical counterpart to equation (8) is $\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{X \in \mathcal{D}_{\text{test}}} \mathcal{E}(k'_\theta, X)$. In our simulation, this empirical counterpart

[18]The larger dispersion of the left panel relative to the right one is in part due to choosing $\sigma_{\text{train}} > \sigma$ to increase exploration.

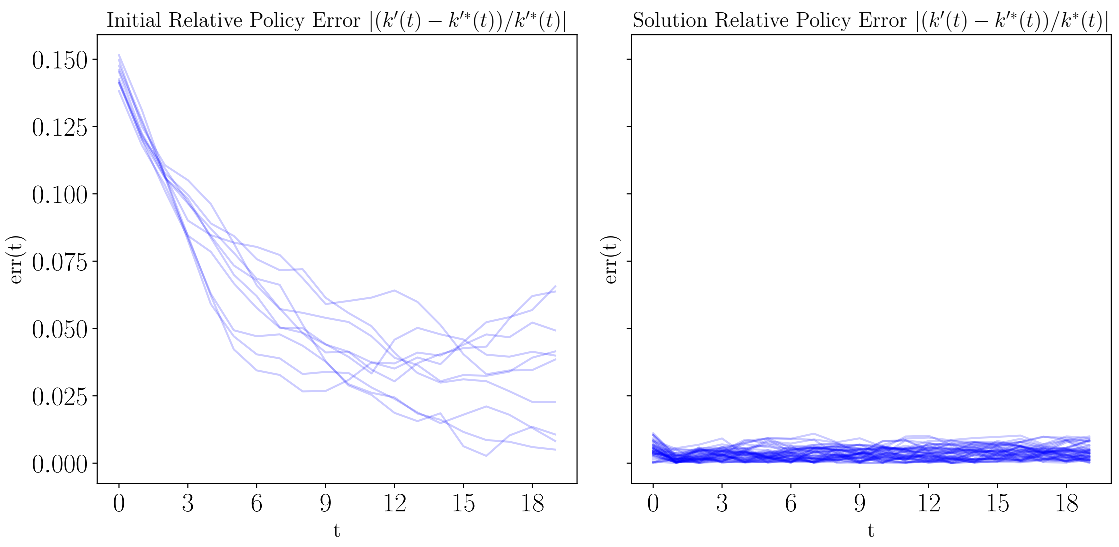

Figure 2: Initial vs. final policy errors

to the population risk is 3.87×10^{-6}, close to numerical precision (the loss of the ERM objective on the final simulated sample is 2.59×10^{-6}).

Furthermore, given that we have a reference solution, $k'_{\text{ref}}(\cdot)$ computed with an extremely accurate conventional policy function iteration, we can calculate $\varepsilon_{\text{rel}} \equiv \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{X \in \mathcal{D}_{\text{test}}} \left| \frac{k'(X) - k'_{\text{ref}}(X)}{k'_{\text{ref}}(X)} \right|$. The absolute relative error of the approximation on $\mathcal{D}$ is 0.0036, and the relative error on the final $\mathcal{D}$ is 0.0046. That is, on average, the policy error is less than half a percent for both the training and the test sets, and it performs better on average for the test set, suggesting that overfitting is not a major concern.

Finally, we check the transversality condition, which may not be fulfilled even if the Euler errors are minimized. To do this, we use 20 simulated trajectories up to $T = 100$ to calculate the expectation in equation (5). See Ebrahim Kahou et al. (2024) for more details on deep learning and transversality conditions.

Other exercises. Another possible exercise of interest could be to estimate the structural parameters, $\{\beta, \alpha, \delta, \rho, \sigma\}$, and possibly the innovations to the productivity shock, $\{\nu_t\}_{t=0}^{T}$, using observed data from the real world. The general approach is to solve for the EMP for a given set of parameters and then match empirical moments or calculate a likelihood that an empirical trajectory would be generated from that EMP. See, for a detailed exposition, Fernández-Villaverde et al. (2016). Alternatively, one can directly perturb the process that generates the EMP, a common approach in Bayesian estimation, especially with gradient-based samplers. See Childers et al. (2022), for example. We skip this exercise for space considerations.

5 Applications of deep NNs in DEMs

Armed with our knowledge of deep learning, we now explore how it can be applied to quantitative economics. Due to length constraints, we will be selective in our coverage, leaving out many most valuable papers.

5.1 Dynamic programming

As we discussed above, the traditional numerical techniques for solving dynamic programming methods (i.e., value function iteration) face the curse of dimensionality. As the number of dimensions increases, the number of elements of the problem explodes, irrespective of whether these elements are just the entries in a high-dimensional table or the coefficients in a multidimensional Chebyshev polynomial. In practice, dealing with problems with a dimensionality higher than four becomes extremely costly. Deep learning promises to solve this problem, opening the door to larger and more realistic optimization problems.

Discrete-time problems. First, we consider the case of discrete dynamic programming. We are interested in solving the Euler equation:

$$\mathbb{E}_t \left[h \left(f \left(x_t \right), f \left(x_{t+1} \right) \right) \right] = 0, \tag{9}$$

where $h\left(\cdot\right)$ is a potentially nonlinear function, x_t is an N-dimensional vector of state variables, and $f(x)$ is a D-dimensional vector of policy functions, mapping from the state variables into policies.

Maliar et al. (2021) and Azinovic et al. (2022) pioneered approximating the policy functions $f\left(\cdot\right)$ by an NN with parameters θ: $\hat{f}\left(x_t;\theta\right)$, the approach in our application in the previous section. The NN is then trained to minimize the Euler equation error:

$$\min_{\theta} \sum_{l=1}^{L} \left[h \left(\hat{f}\left(x_{l,t};\theta\right), \hat{f}\left(x_{l,t};\theta\right) \right) - h\left(f\left(x_{l,t}\right), f\left(x_{l,t+1}\right) \right) \right]^2,$$

over a set of L simulated samples $\{x_{l,t}\}_{l=1}^{L}$. The simulated samples are typically drawn from the ergodic distribution by simulating the model under the approximated policy and the iterative approach described in Section 4. In this way, the accuracy of the algorithm improves in the region of the state space where the solution actually “lives,” which is critical in high-dimensional spaces in which the ergodic distribution typically occupies a very small volume of the state space.

This algorithm is flexible and simple to implement. Variations of it have been applied to solve models of saving-consumption decisions (Gorodnichenko et al., 2020), monetary economics (Nuño et al., 2024), climate change (Folini et al., 2024), and asset pricing (Azinovic and Zemlicka, 2024). Its main shortcoming, however, lies in the computation of expectations. While in low-dimensional spaces, expectations can be easily computed by numerical integration; as dimensionality increases, this task becomes progressively burdensome. Maliar et al. (2021) propose the “all-in-one” expectation operator for an efficient approximation of integrals in Monte Carlo simulation. Another alternative is to move to a continuous-time setting (see below).

Duffy and McNelis (2001) and Villa and Valaitis (2024) propose a different approach. Instead of approximating the inner part of the expectation operator, they approximate the operator itself, in the spirit of the parameterized expectations algorithm introduced by den Haan and Marcet (1990). As discussed by Maliar and Maliar (2003), the main drawback of this approach is that if the assumed decision rule is far from the true solution, the algorithm is likely to diverge.

Continuous-time models. The complication of handling the expectation in equation (9) suggests moving the dynamic optimization problem to continuous time, where the Hamilton-Jacobi-Bellman (HJB) equation can be evaluated without the need to compute any expectations.

For instance, suppose that states x_t follow a diffusion process:

$$dx_t = \mu\left(x_t, c_t\right) dt + \sigma\left(x_t\right) dz_t,$$

where $\mu(\cdot)$ is the drift, $\sigma(\cdot)$ is the diffusion, c_t are the policies, and z_t is a brownian motion. If the agent has an objective function $\int_0^\infty e^{-\rho t} u\left(c_t, x_t\right) dt$, the associated HJB is simply

$$\rho V\left(x\right) = \max_c u\left(c, x\right) + \sum_{n=1}^{N} \mu_n\left(x, c\right) \partial_{x_n} V\left(x\right) + \sum_{n_1, n_2=1}^{N} \frac{\sigma^2_{x_{n_1}, x_2}\left(x\right)}{2} \partial^2_{x_{n_1}, x_2} V\left(x\right),$$

where $V(\cdot)$ is the value function.

No expectations are needed in the HJB since, due to Itô's lemma, the last term of the equation captures all the uncertainty linked to the diffusion process. Nothing substantial would change if we had Poisson jumps in addition to the diffusion process.

The approach proposed by Fernández-Villaverde et al. (2020), Gopalakrishna (2021), Sauzet (2021), and Duarte et al. (2024) is to approximate the value function by an NN $\widehat{V}(x;\theta)$ and to train it to minimize the HJB error:

$$\min_{\theta} \sum_{l=1}^{L} \Big[-\rho \widehat{V}(x_l) + u(c(x_l), x_l) + \sum_{n=1}^{N} \mu_n(x_l, c(x_l))\, \partial_{x_n} \widehat{V}(x_l) + \sum_{n_1,n_2=1}^{N} \frac{\sigma^2_{x_{n_1},x_2}(x)}{2} \partial^2_{x_{n_1},x_2} \widehat{V}(x_l) \Big]^2.$$

The optimal policy $c(\cdot)$ can either be directly derived from the first-order condition (FOC) $\sum_{n=1}^{N} \partial_c \mu_n(x,c)\, \partial_{x_n} \widehat{V}(x) = 0$ or can be approximated by a second NN trained to minimize the error of the first-order condition.[19]

A second advantage of working in continuous time is that the computation of the partial derivatives of the value function is relatively straightforward, thanks to backpropagation. One important detail, however, is that the NN should incorporate the different boundary conditions in the loss function.

A different approach is followed by Huang (2023). Instead of working with the HJB, Huang formulates the problem as a system of forward-backward stochastic differential equations. While he also approximates the value function using an NN, the loss function is evaluated on the path of samples of the simulated value function.

5.2 Heterogeneous-agent models

There is a plethora of computational methods to solve DEMs with heterogeneous agents (i.e., agents that differ from each other in non-trivial ways) under perfect-foresight conditions or subject to aggregate shocks up to a first-order approximation (e.g., Ahn et al., 2018, Boppart et al., 2018; Winberry, 2018; Auclert et al., 2021) or even a second-order (Bhandari et al., 2023).

[19]There is a much longer literature on the use of deep learning to solve partial differential equations (PDEs), not necessarily HJB equations, that we do not revisit here. The interested reader may check, for instance, Han et al. (2018) and Sirignano and Spiliopoulos (2018).

The problem becomes more challenging when trying to solve globally a heterogeneous-agent DEM with aggregate shocks. In that case, the distribution of idiosyncratic states G_t is one state variable in the agent's dynamic programming problem because it determines aggregate prices and other equilibrium conditions. Thus, to solve these models, we need to deal with G_t and the operator $H(\cdot)$ that characterizes how G_t evolves:

$$G_{t+1} = H(G_t, S_t)$$

in discrete time or

$$\frac{\partial G_t}{\partial t} = H(G_t, S_t)$$

in continuous time, given the other aggregate states of the economy S_t. Since G_t is infinite-dimensional and stochastic, it cannot be dealt with using standard (finite-dimensional) calculus rules. Also, the operator $H(\cdot)$ is an equilibrium object: G_t is determined by the MDP of the agents and the EMP. For example, in a DEM of wealth heterogeneity, the agent's wealth in the next period depends on its current wealth, its current optimal choices, aggregate prices, and stochastic shocks.

Krusell and Smith (1998) propose a bounded-rationality solution for a model with a continuum of atomistic agents. When making decisions, these agents summarize G_t by a number of moments and, then, forecast the law of motion of these moments. An equilibrium in this model is a fixed point in which the moments' perceived law of motion (PLM) coincides with the actual law of motion (ALM) observed along simulated paths. The question is how to build the PLM.

In Krusell and Smith (1998), a (log-)linear PLM with state-dependent coefficients provided a good approximation to the ALM in the baseline neoclassical model with incomplete markets and aggregate shocks, as the model is quite linear. There is no guarantee, however, that a linear PLM also provides a good approximation to models with more prominent nonlinearities at the macroeconomic level. Once we abandon the linear world, there are infinite possible functional forms. Paraphrasing Tolstoy: "Linear models are all alike; every nonlinear model is nonlinear in its own way."

Approximating the PLM. Fernández-Villaverde et al. (2023) propose a solution to this problem by approximating the PLM by an NN and training it using simulated data from the model. While Fernández-Villaverde et al. (2023) work in continuous time, Fernández-Villaverde et al. (2024) extend the idea to discrete-time settings.

More concretely, Fernández-Villaverde et al. (2023) propose extracting a finite number of features from G_t. These can be moments, Q-quantiles, weights in a mixture of normals, or many other options. Then, one can stack the features of the distribution in a vector μ_t and assume that μ_t follows a PLM:

$$\mu_{t+1} = h(\mu_t, S_t)$$

in discrete time or

$$\frac{\partial \mu_t}{\partial t} = h(\mu_t, S_t)$$

in continuous time, instead of the exact $H(G_t, S_t)$. We parameterize $h(\mu_t, S_t)$ as $h(\mu_t, S_t; \theta)$ using an NN, where θ is the vector of network weights. We determine the unknown weights θ to ensure that an economy where μ_t follows $h(\mu_t, S_t; \theta)$ replicates as well as possible the behavior of an economy where G_t follows $H(\cdot)$. To do so:

1. We guess an initial value θ^0 of the NN weights. We set $n = 0$.

2. We construct a time series of μ_t simulating from the PLM and the relevant equilibrium conditions of the model.

3. We train the NN weights on the simulated data and obtain θ^{n+1} by minimizing a quadratic error function between μ_{t+1} and μ_t (in discrete time) or $\frac{\mu_{t+1}-\mu_t}{\Delta t}$ and μ_t (in continuous time).

4. We iterate on steps 2-3 until $||\theta^{n+1} - \theta^n|| < \epsilon$ given a norm $||\cdot||$ and tolerance level $\epsilon > 0$.

This algorithm nests the approach of Krusell and Smith (1998) as a particular case since the linear formulation that the authors pick for the PLM is just a simple NN.

Deep HAM. One shortcoming of the Fernández-Villaverde et al. (2023) approach is that the researcher must select the relevant moments $h(\mu_t, S_t)$ based on the economics of the problem. Instead, Han et al. (2022) propose approximating the state distribution by a set of optimal generalized moments. NNs represent these generalized moments, which are automatically determined by the algorithm. Deep NNs are also used to approximate the value and policy functions, and the objective is optimized over directly simulated paths, as in the methods discussed in the previous subsection.

Interestingly, Han et al. (2022) pick an objective function based on cumulated utility instead of first-order conditions, which makes DeepHAM useful for solving models that involve complex first-order conditions, e.g., constrained efficiency problems such as in Dávila et al. (2012). This also makes DeepHAM share a similar structure with model-based reinforcement learning.

Approximating the master equation. The previous methods reduce the operator $H(\cdot)$ to a vector of moments $h(\mu_t, S_t)$. One alternative is to deal directly with the operator via the "master equation" in continuous time (Bilal, 2023). The master equation summarizes both the agent optimization behavior from the Hamilton-Jacobi-Bellman equation and the evolution of the distribution from the Kolmogorov forward equation. Gu et al. (2023) approximate the infinite-dimensional master equation by a finite but high-dimensional PDE and then use deep learning to solve the high-dimensional equation. They consider two different approaches for reducing the dimension of the master equation. The first approach approximates the distribution by a large, finite number of agents. The second approach approximates the distribution by discretizing the agent state space so the density becomes a collection of mass points at grid points.

This methodology has been applied to search and matching models (Payne et al., 2024) and asset pricing (Gopalakrishna et al., 2024).

Exploiting symmetry. Ebrahimi Kahou et al. (2021) propose a method for solving models with a large but finite number of heterogeneous agents using deep learning. This is different from the problems described so far, which concern a continuum of atomistic agents. Still, both cases are related as the traditional heterogeneous-agent models can be seen as a limit of a large, finite number of agents. Ebrahimi Kahou et al. (2021) tame the curse of dimensionality by exploiting the symmetry in the PLM and using it to engineer an efficient representation of the state space.

5.3 Reinforcement learning

Reinforcement learning is a growing field in computer science and robotics (Sutton and Barto, 2018). It deals with dynamic programming problems in which the state dynamics are unknown, and, hence, agents need to learn them by observing how states evolve under the chosen policies. Some of the recent breakthroughs mentioned in the introduction are based on deep reinforcement learning, that is, the application of deep NNs to reinforcement learning problems (Jaderberg et al., 2016, and Silver et al., 2016). This typically involves approximating the value and policy functions by NNs.

In macroeconomics, deep reinforcement learning methods have been employed to model learning in complex nonlinear environments. In that respect, deep reinforcement learning provides a generalization to the least-squares learning methods (e.g., Marcet and Sargent, 1989, or Evans and Honkapohja, 2001). Charpentier et al. (2020) and Atashbar and Shi (2022) review the application of deep and reinforcement learning in finance and economics. Here, we highlight a few papers. Covarrubias (2023) evaluates the effect of dynamic oligopolistic strategies on the transmission of monetary policy by solving the strategic problem of firms using deep reinforcement learning. Atashbar and Shi (2022) employ deep reinforcement learning to solve a real business cycle model. Chen et al. (2023) solve a monetary model. Finally, Hinterlang and Tänzer (2021) employ deep reinforcement learning to compute optimal monetary policy.

5.4 Estimation

NNs can also be used to estimate more efficiently DEM models. This is a computationally intensive task that requires solving the model for a large number of parameter values and then evaluating the likelihood using a sequential Monte Carlo filter (Fernández-Villaverde and Guerrón-Quintana, 2021).

Kase et al. (2024) propose a new method to speed up estimation in the context of heterogeneous-agent New Keynesian (HANK) models with aggregate shocks. First, they compute the equilibrium using deep learning as proposed by Maliar et al. (2021) and discussed in Section 5.1.

Second, they treat the parameters as pseudo-state variables by exploiting the scalability of NNs. Then, they train this extended NN with the parameters as inputs over the entire parameter space. This step yields the mapping from the parameter space to the model's equilibrium law of motion instead of just one solution for a single point in the parameter space. Even though treating the model's parameters as pseudo-state variables makes it more challenging to train the NN, the computational gains are large compared to repeatedly evaluating the model's solution mapping at as many points as needed to get an appreciable evaluation of the likelihood function.[20]

Third, they train an additional NN that provides a direct mapping from the model parameters to the value of the likelihood function. For this strategy, they evaluate the likelihood using the particle filter for different parameter combinations over the

[20]A similar approach, dubbed "deep surrogates," is employed in finance by Chen et al. (2021).

parameter space and use these as data points to train this second NN. Because of the good generalization properties of NNs, this training requires only thousands of data points, thus reducing the computation time significantly compared to conventional estimation, which typically requires millions of draws.

A related idea appears in Friedl et al. (2023), who solve the model as a function of its states and parameters in one go using deep NNs. Instead of simulating the moments, however, the authors simulate the social cost of carbon, and then compute the Sobol indices, univariate effects, and Shapley values for global uncertainty quantification.

6 Traditional methods vs. NNs

In Section 3, we saw there is a key tradeoff between the parsimony of basis functions and the rich parameterization of $\theta_{n,m}$ in equation (2) in comparison with equation (3). This point is actually more general and carries over to other ML methods. Recall that ML includes different tools to fill the two large volumes of Murphy (2022, 2024), from OLS to transformer models for LLMs. Thus, it is helpful to characterize five distinguishing features of modern ML, including deep NNs, more in general.

Distinguishing feature 1: Overparameterization. Classic function approximations, such as equation (3), usually choose approximations with fewer parameters than there are data to avoid overfitting. In comparison, ML algorithms, such as equation (2), often have orders of magnitude more parameters than data. Even more strikingly, kernel methods solve a problem in an infinite-dimensional feature space.

However, simply counting parameters might provide the wrong metric when gauging an approximation or its relationship with the amount of data (Maddox et al., 2020, and Curth et al., 2023). Computer scientists and statisticians usually refer to the flexibility of the functional form as "capacity," which generalizes the notion of counting parameters to relate better to notions of complexity.

From the perspective of quantitative economics, since we are using overparameterized approximations with a great deal of capacity, we can fit complicated functions of the very high-dimensional state spaces of complex DEMs. In particular, we will have solutions that fit the model structure at all grid points to numerical precision and not trade off bias and variance. The double descent phenomenon (to be introduced momentarily) suggests this would not lead to catastrophic overfitting.

Distinguishing feature 2: Regularization. Since ML problems are overparameterized, one must use various forms of regularization to guide algorithms in choosing solutions with desirable properties. A well-known example is LASSO, where regularization biases toward solutions with fewer parameters. Similarly, kernel methods regularize the solution in a function space as part of a representer theorem. With deep NNs, regularization may occur in subtle ways, such as the design of the NN, from the optimization algorithm itself, or the noise in the data (see Barrett and Dherin, 2020, and Smith et al., 2021).[21] As we mentioned above, deep NN will usually not pick parameters like classical projection methods (e.g., using a collocation or similar), but by minimizing an appropriately regularized loss function.[22]

Following our notation a few pages ago, regularization will ensure that we do not identify spurious features in our underlying transformation of the state through the $\phi(X)$ and that we do not overfit to the output through the $g(\cdot)$.

In fact, there is an even more fundamental issue at play here. Highly overparameterized models can induce an important source of regularization in what is called "double descent." This is the phenomenon where the performance of the ML approximation, for instance, as measured in terms of the loss function (4):

1. First improves as we increase the total number of parameters, and the model learns the data ("first descent").
2. Then, deteriorates as we overfit with too many parameters, i.e., the model simply "memorizes" the data.
3. But, next, after we cross the interpolation threshold by adding even more parameters, the fit improves again ("second descent").

Thus, highly overparameterized models seem to escape the bias-variance tradeoff. While the double descent phenomenon has been documented at least since Vallet et al. (1989), researchers are still struggling to grasp its origin and implications (Belkin

[21]To give one example of regularization in this context: auto-encoders are a classic method to find $\phi(\cdot)$ by attempting to fit the data with both $\phi(\cdot)$ and an approximation for $\phi^{-1}(\cdot)$. In that case, a key source of regularization is to *add* noise to the data (Kingma and Welling, 2013).

[22]Also, the fact that there might be different combinations of parameter values that provide the same answer to equation (4) is not a concern: all these solutions will deliver the same approximation function. One must avoid overfacile analogies with econometrics, where point identification is often a key desirable property. In comparison, for solving DEM, point identification of the θ's is irrelevant.

et al., 2019, and Belkin, 2024). In econometrics, Spiess et al. (2023) provide recent examples of double descent in causal inference.

Distinguishing feature 3: Representations. A common element of ML is a transformation of the underlying data into features that may be more amenable to the task. This might involve artful "engineering" (e.g., manually transforming data with first differences). But, as we described when talking about the geometrical interpretation of deep NNs, this could involve learning a transformation from patterns in the structure of the data (as occurs when fitting NN, auto-encoder-based algorithms, and dimension-reduction techniques). In terms of our goals, ML approximations can often transform the state space into an efficient representation that captures the key payoff-relevant features of the DEM.

Distinguishing feature 4: Concentration of measure. Many ML methods encourage different approximation errors in different regions, with an emphasis on the "typical sets" of the state variables, which become increasingly representative (and of smaller relative volume) of the space due to the concentration of measure phenomena (Ledoux, 2001).[23]

In some DEMs, the high-dimensional states provide a blessing of dimensionality rather than a curse: the typical sets shrink dramatically because of concentration of measure (one can think about this concentration as a law of large numbers for endogenous equilibrium objects).

The key change in perspective is that dimensionality, either in the state or parameter space, is not inherently a problem. In some cases, dimensionality is even a blessing rather than a curse—for example, having more samples of data helps estimation, and either going to a continuum limit as in mean-field games or large numbers of discrete agents helps simplify equilibrium models.

In general, dimensionality is something neither to fear nor necessarily to conquer, and solutions are always problem-dependent. In that sense, as long as we are willing to

[23]Loosely speaking, the typical set is the region of the space of interest that matters the most for the computation of objects of interest such as conditional expectations. The geometry of typical sets is often counterintuitive, particularly in high dimensions. See Fernández-Villaverde and Guerrón-Quintana (2021) for an introduction to typical sets and their properties and `https://betanalpha.github.io/assets/case_studies/probabilistic_computation.html` for an introduction to Hamiltonian Monte Carlos designed to sample from typical sets efficiently.

ignore the worst case and define success probabilistically, the curse of dimensionality is not a fundamental barrier.

Distinguishing feature 5: High-dimensional optimization. The ML toolkit leverages high-dimensional optimization algorithms that can solve problems with many parameters and regularize non-convex objectives. Advances in software to find gradients of arbitrary functions and hardware to implement them are key features of ML frameworks such as `Tensorflow`, `Pytorch`, and `JAX`. These advances make the optimization of loss functions associated with DEMs feasible. A crucial ingredient of these methods is the ability to calculate high-dimensional gradients of the functions implied by DEMs (Blondel and Roulet, 2024).

We conclude this section by noting that we have deliberately downplayed big data in our discussion. While increasing data can be beneficial —especially when learning representations —it is neither necessary nor sufficient for success.[24] The impact of large amounts of data is nuanced, and intuition based on low-dimensional identified systems often fails. For example, additional data may improve generalization primarily due to its noise, which enhances regularization (see Bishop, 1995 and Hastie et al., 2022). Conversely, increasing data while keeping the number of parameters fixed can sometimes undermine double descent, leading to poorer performance.

7 Reasons for cautious optimism

Throughout the paper, we have emphasized that the success of deep NNs is due to the change of geometry in the representation state. Our goal now is to understand better how this happens by presenting three reasons for cautious optimism.

7.1 Reason 1: Low-dimensional representations

Several methods in economics have successfully handled some high-dimensional problems. First, perturbation works because (i) it focuses on a small region of the state space (i.e., the region around the deterministic steady state), and (ii) imposes the

[24] Ebrahimi Kahou et al. (2021) solve a model reasonably accurately with hundreds of thousands of parameters and only three points in 128-dimensional space. This is possible because the representation space $\mathcal{Z}$ enforces structure, regularization is strong, and the states are concentrated within the state space when conditioned on an initial condition.

condition that the solutions be "stable" and will not overfit. Second, we have solutions in dynamic IO —such as the self-confirming equilibria in Fudenberg and Levine (1993) and the related experience-based equilibria in Fershtman and Pakes (2012)— that emphasize smaller, empirically driven regions of the state space where less structure is imposed on off-equilibrium beliefs. That is, the MDP is solved with minimal structure on the states outside of the empirical draws from the EMP. Third, Weintraub et al. (2008) address the curse of dimensionality by transforming the state space into a low-dimensional variable that represents long-run dynamics.

The results in Ebrahimi Kahou et al. (2021) and Ebrahim Kahou et al. (2024) suggest that deep learning replicates the reasons why the methods above work. For example, based on the exploitation of symmetry in Ebrahimi Kahou et al. (2021), we conjecture that deep learning methods can learn the low-dimensional representations that the three methods above implicitly exploit. The authors also document that the problems can be solved with fewer ad-hoc sources of sub-optimality for the "off-equilibrium" beliefs and best responses. Ebrahim Kahou et al. (2024) discuss the tight connection between regularization in deep learning and stability that arises from the inductive bias built into many ML methods.

The tradeoff behind these ML methods is that some regions of the state space (which, ideally, the EMP reaches with low probability for a given counterfactual experiment) will be poorly approximated. In cases where nearly all of the state space is relevant, either for the EMP itself or the off-equilibrium considerations of the MDP, the curse of dimensionality is unavoidable.

More generally, economists are trained to be cautious with high dimensionality in both the state space and the number of parameters, remaining justifiably skeptical of claims that it can be easily overcome. However, since most ML methods transform the state into an engineered or learned representation, the key dimensionality of interest is that of the representation, not the original space. For instance, if the function to approximate is a nonlinear mapping of the data's mean (as in the PLMs of many heterogeneous-agent models), the mean itself becomes the ideal representation.

7.2 Reason 2: The manifold hypothesis

As briefly mentioned above, the "manifold hypothesis" conjectures that real-world data are always concentrated on a low-dimensional manifold (Fefferman et al., 2016).

That is, the world is much simpler than it appears, though what qualifies as "simple" may vary based on the goal. This manifold hypothesis arises from the interaction between a concentration of measure, where reasonable stochastic processes in high dimensions lead to concentration within a small volume of space, and representations that compress the payoff-relevant information of a high-dimensional state space into a much lower-dimensional one.

The manifold hypothesis may explain the remarkable success of deep learning in solving complex problems in computer vision and natural language processing, such as ChatGPT. From this view, the structure of real-world images or text is much simpler than it seems. For instance, as Belkin (2024) notes, GPT-3.5 turbo is trained to predict the next token among roughly 10^4 possible tokens, within a context window of 4,000. Even with $\sim 10^{12}$ data points, naively estimating an unstructured transition matrix would resemble reconstructing a library from a molecule of ink. In contrast, the inherent low-dimensional structure of data makes learning feasible.

Suppose the manifold hypothesis seems to be true with real-world data. In that case, it should also be true with DEMs, which are constructed by researchers to be relatively simple, even if the economist often does not know the most useful representation of the model.

7.3 Reason 3: Transferability between models and invariants

ML can help us reuse calculations across versions of a DEM, such as when estimating structural parameters or performing comparative statics. This transferability arises from using deep representations to approximate the state space.

Good representations are often reusable across different tasks, as much of the work in fitting an NN involves finding a strong representation of the state. Once such a representation is established, only the final "layer" in the nested function approximation needs adjusting to suit a new task. Components of the approximation can then be used as an initial condition or kept fixed while the rest of the model is resolved. This is convenient with deep NNs, which allow for "model surgery" —the dissection of pre-trained layers to fix some nested approximations while leaving others to adapt. This process is known as "transfer learning" (Zhuang et al., 2020).

Specifically, we can use a $\phi(\cdot)$ function from a prior problem and combine it with a new function $g(\cdot)$ to form $f(X) \approx g(\phi(X))$. When employing differentiable programming, it becomes practical to compute how an entire EMP might change with respect to a deep parameter. For an introduction to differentiable programming, see Blondel and Roulet (2024), and for a DEM estimation example, see Childers et al. (2022).

8 A glimpse into the future

We are only beginning to see the full impact of the artificial intelligence revolution sparked by Krizhevsky et al. (2012). Nevertheless, its effects are already pervasive in daily life, from self-driving cars taking us to school in the morning to LLMs drafting emails to the dean's office. Quantitative economics is no exception.

For decades, economists have worked under significant constraints on the types of models they could address, often focusing on models they could solve rather than those they wanted to solve. Deep NNs, when applied judiciously to tame the curse of dimensionality, can expand the scope of our work in ways that were hard to imagine a few years ago. However, this is not due to "magic" but to better geometric representations of the state space. Much effort is still needed to understand representation theory, related concepts like the double descent phenomenon, and their implications for economics. Finally, a key advantage of deep NNs is their compatibility with a new generation of hardware that drastically expands the class of models we can compute (Fernández-Villaverde and Valencia, 2018; Cheela et al., 2022).

These advances hold transformative implications for economics and economic policy. For example, Fernández-Villaverde et al. (2025) review how deep NNs could reshape the models used to analyze the macroeconomic impacts of climate change, while Carvalho et al. (2024) employ deep NNs to study the nonlinear propagation of sectoral shocks. Also, the shift in quantitative economics toward microdata-driven models often necessitates high-dimensional state spaces, underscoring the potential of deep learning in such contexts. To echo Howard Carter's famous response on November 26, 1922, when asked by Lord Carnarvon if he could see anything as he peered into King Tutankhamun's tomb, we can also answer about what we see in the future of quantitative economics: "Yes, wonderful things!"

References

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

Atashbar, T. and R. A. Shi (2022): "Deep reinforcement learning: Emerging trends in macroeconomics and future prospects," Working Paper 2022/259, IMF.

Athey, S. and G. W. Imbens (2019): "Machine learning methods that economists should know about," *Annual Review of Economics*, 11, 685–725.

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

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

Azinovic, M. and J. Zemlicka (2024): "Intergenerational consequences of rare disasters," Manuscript.

Bach, F. (2017): "Breaking the curse of dimensionality with convex neural networks," *Journal of Machine Learning Research*, 18, 629–681.

Barrett, D. G. and B. Dherin (2020): "Implicit gradient regularization," Tech. Rep. 2009.11162, arXiv.

Barron, A. (1993): "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, 39, 930–945.

Belkin, M. (2024): "The puzzle of dimensionality and feature learning from LLMs to kernel machines," Plenary lecture at the 24th ACM Conference on Economics and Computation (ACM-EC).

Belkin, M., D. Hsu, S. Ma, and S. Mandal (2019): "Reconciling modern machine-learning practice and the classical bias–variance trade-off," *Proceedings of the National Academy of Sciences of the United States of America*, 116, 15849–15854.

Bellman, R. (1957): *Dynamic Programming*, Princeton University Press.

Benkard, C. L., P. Jeziorski, and G. Y. Weintraub (2015): “Oblivious equilibrium for concentrated industries,” *RAND Journal of Economics*, 46, 671–708.

Bhandari, A., T. Bourany, D. Evans, and M. Golosov (2023): “A perturbational approach for approximating heterogeneous agent models,” NBER Working Papers 31744, National Bureau of Economic Research.

Bilal, A. (2023): “Solving heterogeneous agent models with the master equation,” NBER Working Papers 31103, National Bureau of Economic Research.

Bishop, C. M. (1995): “Training with noise is equivalent to Tikhonov regularization,” *Neural Computation*, 7, 108–116.

Blondel, M. and V. Roulet (2024): “The elements of differentiable programming,” Tech. Rep. 2403.14606, arXiv.

Boppart, T., P. Krusell, and K. Mitman (2018): “Exploiting MIT shocks in heterogeneous-agent economies: The impulse response as a numerical derivative,” *Journal of Economic Dynamics and Control*, 89, 68–92.

Bronstein, M. M., J. Bruna, T. Cohen, and P. Veličković (2021): “Geometric deep learning: Grids, groups, graphs, geodesics, and gauges,” Tech. Rep. 2104.13478, arXiv.

Brumm, J. and S. Scheidegger (2017): “Using adaptive sparse grids to solve high-dimensional dynamic models,” *Econometrica*, 85, 1575–1612.

Carvalho, V., M. Covarrubias, and G. Nuño (2024): “Nonlinearities and amplification in dynamic production networks,” Manuscript.

Cayton, L. (2005): “Algorithms for manifold learning,” Technical report, University of California at San Diego.

Chakraborty, C. and A. Joseph (2017): “Machine learning at central banks,” Working paper 674, Bank of England.

Charpentier, A., R. Elie, and C. Remlinger (2020): “Reinforcement learning in economics and finance,” Tech. Rep. 2003.10014, arXiv.

Cheela, B., A. DeHon, J. Fernández-Villaverde, and A. Peri (2022): "Programming FPGAs for economics: An introduction to electrical engineering economics," Working Paper 29936, National Bureau of Economic Research.

Chen, H., A. Didisheim, and S. Scheidegger (2021): "Deep surrogates for finance: With an application to option pricing," Available at SSRN 3782722.

Chen, M., A. Joseph, M. Kumhof, X. Pan, and X. Zhou (2023): "Deep reinforcement learning in a monetary model," Tech. Rep. 2104.09368, arXiv.

Chiang, P.-y., R. Ni, D. Y. Miller, A. Bansal, J. Geiping, M. Goldblum, and T. Goldstein (2022): "Loss landscapes are all you need: Neural network generalization can be explained without the implicit bias of gradient descent," in *The Eleventh International Conference on Learning Representations*.

Childers, D., J. Fernández-Villaverde, J. Perla, C. Rackauckas, and P. Wu (2022): "Differentiable state space models and Hamiltonian Monte Carlo estimation," Working Paper 30573, National Bureau of Economic Research.

Chollet, F., T. Kalinowski, and J. Allaire (2022): *Deep Learning with R*, Manning Publications, 2nd ed.

Covarrubias, M. (2023): "Dynamic oligopoly and monetary policy: A deep reinforcement learning approach," Manuscript.

Cover, T. M. (1965): "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Transactions on Electronic Computers*, EC-14, 326–334.

Curth, A., A. Jeffares, and M. van der Schaar (2023): "A u-turn on double descent: Rethinking parameter counting in statistical learning," Tech. Rep. 2310.18988, arXiv.

Cybenko, G. (1989): "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, 2, 303–314.

Dávila, J., J. H. Hong, P. Krusell, and J.-V. Ríos-Rull (2012): "Constrained efficiency in the neoclassical growth model with uninsurable idiosyncratic shocks," *Econometrica*, 80, 2431–2467.

de Araujo, D. K. G., S. Doerr, L. Gambacorta, and B. Tissot (2024): "Artificial intelligence in central banking," BIS Bulletins 84, Bank for International Settlements.

Dell, M. (2024): "Deep learning for economists," *Journal of Economic Literature*, forthcoming.

den Haan, W. J. and A. Marcet (1990): "Solving the stochastic growth model by parameterizing expectations," *Journal of Business & Economic Statistics*, 8, 31–34.

Duarte, V., D. Duarte, and D. Silva (2024): "Machine learning for continuous-time finance," *Review of Financial Studies*, 11, 3217–3271.

Duffy, J. and P. D. McNelis (2001): "Approximating and simulating the stochastic growth model: Parameterized expectations, neural networks, and the genetic algorithm," *Journal of Economic Dynamics and Control*, 25, 1273–1303.

Ebrahim Kahou, M., J. Fernández-Villaverde, S. Gomez-Cardona, J. Perla, and J. Rosa (2024): "Spooky boundaries at a distance: Inductive bias, dynamic models, and behavioral macro," Working Paper 32850, National Bureau of Economic Research.

Ebrahimi Kahou, M., J. Fernández-Villaverde, J. Perla, and A. Sood (2021): "Exploiting symmetry in high-dimensional dynamic programming," Working Paper 28981, National Bureau of Economic Research.

Evans, G. W. and S. Honkapohja (2001): *Learning and Expectations in Macroeconomics*, Princeton University Press.

Fefferman, C., S. Mitter, and H. Narayanan (2016): "Testing the manifold hypothesis," *Journal of the American Mathematical Society*, 29, 983–1049.

Fernández-Villaverde, J., K. Gillingham, and S. Scheidegger (2025): "Climate change through the lens of macroeconomic modeling," *Annual Review of Economics*, Forthcoming.

Fernández-Villaverde, J. and P. A. Guerrón-Quintana (2021): "Estimating DSGE models: Recent advances and future challenges," *Annual Review of Economics*, 13.

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

Fernández-Villaverde, J., J. Marbet, G. Nuño, and O. Rachedi (2024): "Inequality and the zero lower bound," *Journal of Econometrics*, 105819.

Fernández-Villaverde, J., G. Nu no, G. Sorg-Langhans, and M. Vogler (2020): "Solving high-dimensional dynamic programming problems using deep learning," Manuscript.

Fernández-Villaverde, J., J. F. Rubio-Ramírez, and F. Schorfheide (2016): "Solution and estimation methods for DSGE models," in *Handbook of Macroeconomics*, Elsevier, vol. 2, 527–724.

Fernández-Villaverde, J. and D. Z. Valencia (2018): "A practical guide to parallelization in economics," Working Paper 24561, National Bureau of Economic Research.

Fershtman, C. and A. Pakes (2012): "Dynamic games with asymmetric information: A framework for empirical work," *Quarterly Journal of Economics*, 127, 1611–1661.

Folini, D., A. Friedl, F. Kübler, and S. Scheidegger (2024): "The climate in climate economics," *Review of Economic Studies*, rdae011.

Friedl, A., F. Kübler, S. Scheidegger, and T. Usui (2023): "Deep uncertainty quantification: With an application to integrated assessment models," Working paper, University of Lausanne.

Fudenberg, D. and D. K. Levine (1993): "Self-confirming equilibrium," *Econometrica*, 523–545.

Goodfellow, I., Y. Bengio, and A. Courville (2016): *Deep Learning*, MIT Press.

Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014): "Generative adversarial networks," Tech. Rep. 1406.2661, arXiv.

Gopalakrishna, G. (2021): "ALIENs and continuous time economies," Swiss Finance Institute Research Paper Series 21-34, Swiss Finance Institute.

Gopalakrishna, G., Z. Gu, and J. Payne (2024): "Institutional asset pricing, segmentation, and inequality," Manuscript.

Gorodnichenko, Y., S. Maliar, and C. Naubert (2020): "Household savings and monetary policy under individual and aggregate stochastic volatility," Discussion paper 15614, CEPR.

Gu, Z., M. Lauriere, S. Merkel, and J. Payne (2023): "Global solutions to master equations for continuous time heterogeneous agent macroeconomic models," Available at SSRN 4871228.

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

Han, J., Y. Yang, and W. E (2022): "DeepHAM: A global solution method for heterogeneous agent models with aggregate shocks," Tech. Rep. 2112.14377, arXiv.

Hansen, L. P. and T. J. Sargent (1980): "Formulating and estimating dynamic linear rational expectations models," *Journal of Economic Dynamics and Control*, 2, 7–46.

Hastie, T., A. Montanari, S. Rosset, and R. J. Tibshirani (2022): "Surprises in high-dimensional ridgeless least squares interpolation," *Annals of Statistics*, 50, 949.

Hinterlang, N. and A. Tänzer (2021): "Optimal monetary policy using reinforcement learning," Discussion Papers 51/2021, Deutsche Bundesbank.

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

Huang, J. (2023): "Breaking the curse of dimensionality in heterogeneous-agent models: A deep learning-based probabilistic approach," Available at SSRN 4649043.

Jaderberg, M., V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu (2016): "Reinforcement learning with unsupervised auxiliary tasks," Tech. Rep. 1611.05397, arXiv.

Jumper, J. M., R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis (2021): "Highly accurate protein structure prediction with AlphaFold," *Nature*, 596, 583 – 589.

Kaji, T., E. Manresa, and G. Pouliot (2023): "An adversarial approach to structural estimation," *Econometrica*, 91, 2041–2063.

Kase, H., L. Melosi, and M. Rottner (2024): "Estimating nonlinear heterogeneous agent models with neural networks," Research Paper Series 1499, University of Warwick, Department of Economics.

Kelly, B. T. and D. Xiu (2023): "Financial machine learning," NBER Working Papers 31502, National Bureau of Economic Research.

Kingma, D. P. and J. Ba (2017): "Adam: A method for stochastic optimization," Tech. Rep. 1412.6980, arXiv.

Kingma, D. P. and M. Welling (2013): "Auto-encoding variational Bayes," Tech. Rep. 1312.6114, arXiv.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012): "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 1097–1105.

Krusell, P. and A. A. Smith, Jr (1998): "Income and wealth heterogeneity in the macroeconomy," *Journal of Political Economy*, 106, 867–896.

Ledoux, M. (2001): *The Concentration of Measure Phenomenon*, American Mathematical Society.

Maddox, W. J., G. Benton, and A. G. Wilson (2020): "Rethinking parameter counting in deep models: Effective dimensionality revisited," Tech. Rep. 2003.02139, arXiv.

Maliar, L. and S. Maliar (2003): "Parameterized expectations algorithm and the moving bounds," *Journal of Business & Economic Statistics*, 21, 88–92.

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

Marcet, A. and T. J. Sargent (1989): "Convergence of least squares learning mechanisms in self-referential linear stochastic models," *Journal of Economic Theory*, 48, 337–368.

Mitchell, T. M. and T. M. Mitchell (1997): *Machine Learning*, McGraw Hill.

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

Murphy, K. P. (2022): *Probabilistic Machine Learning: An Introduction*, MIT Press.

——— (2024): *Probabilistic Machine Learning: Advanced Topics*, MIT Press.

Nagel, S. (2021): *Machine Learning in Asset Pricing*, Princeton University Press.

Nakkiran, P., G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever (2019): "Deep double descent: Where bigger models and more data hurt," Tech. Rep. 1912.02292, arXiv.

Nuño, S. Scheidegger, and P. Renner (2024): "Let bygones be bygones: Optimal monetary policy with persistent supply shocks," Manuscript.

Payne, J., A. Rebei, and Y. Yang (2024): "Deep learning for search and matching models," Manuscript.

Robbins, H. and S. Monro (1951): "A stochastic approximation method," *Annals of Mathematical Statistics*, 22, 400–407.

ROSENBLATT, F. (1958): "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, 65, 386–408.

RUMELHART, D. E., G. E. HINTON, AND R. J. WILLIAMS (1986): "Learning representations by back-propagating errors," *Nature*, 323, 533–536.

SARGENT, T. J. (2024): "Macroeconomics after Lucas," Working Paper.

SAUZET, M. (2021): "Projection methods via neural networks for continuous-time models," Manuscript.

SCHEIDEGGER, S. AND I. BILIONIS (2019): "Machine learning for high-dimensional dynamic stochastic economies," *Journal of Computational Science*, 33, 68–82.

SHEN, Z., R. ZHANG, M. DELL, B. LEE, J. CARLSON, AND W. LI (2021): "LayoutParser: A unified toolkit for deep learning based document image analysis," *International Conference on Document Analysis and Recognition*, 131–146.

SILVER, D., A. HUANG, C. J. MADDISON, A. GUEZ, L. SIFRE, G. VAN DEN DRIESSCHE, J. SCHRITTWIESER, I. ANTONOGLOU, V. PANNEERSHELVAM, M. LANCTOT, S. DIELEMAN, D. GREWE, J. NHAM, N. KALCHBRENNER, I. SUTSKEVER, T. LILLICRAP, M. LEACH, K. KAVUKCUOGLU, T. GRAEPEL, AND D. HASSABIS (2016): "Mastering the game of Go with deep neural networks and tree search," *Nature*, 529, 484–489.

SIRIGNANO, J. AND K. SPILIOPOULOS (2018): "DGM: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, 375, 1339–1364.

SMITH, S. L., B. DHERIN, D. BARRETT, AND S. DE (2021): "On the origin of implicit regularization in stochastic gradient descent," in *International Conference on Learning Representations*.

SPIESS, J., G. IMBENS, AND A. VENUGOPAL (2023): "Double and single descent in causal inference with an application to high-dimensional synthetic control," NBER Working Papers 31802, National Bureau of Economic Research.

SUTTON, R. S. AND A. G. BARTO (2018): *Reinforcement Learning: An Introduction*, Bradford.

Trinh, T., Y. T. Wu, Q. Le, H. He, and T. Luong (2024): "Solving olympiad geometry without human demonstrations," *Nature*, 625, 476–482.

Vallet, F., J.-G. Cailton, and P. Refregier (1989): "Linear and nonlinear extension of the pseudo-inverse solution for learning Boolean functions," *Europhysics Letters*, 9, 315.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017): "Attention is all you need," in *Advances in Neural Information Processing Systems*, 5998–6008.

Vershynin, R. (2018): *High-Dimensional Probability: An Introduction with Applications in Data Science*, vol. 47, Cambridge University Press.

Villa, A. T. and V. Valaitis (2024): "Machine learning projection method for macro-finance models," *Quantitative Economics*, 15, 145–173.

Voth, H.-J. and D. Yanagizawa-Drott (2024): "Image(s)," Tech. rep., University of Zurich.

Wainwright, M. J. (2019): *High-Dimensional Statistics: A Non-Asymptotic Viewpoint*, vol. 48, Cambridge University Press.

Weintraub, G. Y., C. L. Benkard, and B. Van Roy (2008): "Markov perfect industry dynamics with many firms," *Econometrica*, 76, 1375–1411.

——— (2010): "Computational methods for oblivious equilibrium," *Operations Research*, 58, 1247–1265.

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

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals (2021): "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, 64, 107–115.

Zhuang, F., Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He (2020): "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, 109, 43–76.