Exploiting Symmetry in High-Dimensional Dynamic Programming

Mahdi Ebrahimi Kahou[1] Jesús Fernández-Villaverde[2] Jesse Perla[1] Arnav Sood[3]

October 17, 2022

[1]University of British Columbia, Vancouver School of Economics

[2]University of Pennsylvania

[3]Carnegie Mellon University

Motivation

- Solve heterogenous agent models defined with standard recursive equilibria we know (and love!)
- Many interesting models in macro/IO/trade/etc. have a finite number of agents:
  - Many locations (countries, regions, metropolitan areas, industries) and even networks.
  - Industry dynamics with many firms and industries.
  - Even bread-and-butter heterogeneous agent labor models (e.g., overlapping generations, different types)
- "Continuum" approximations—when feasible—have difficulty with aggregate shocks
- **No (non-heuristic) algorithms exist** for global heterogeneous agent models with aggregate shocks
  - Krusell-Smith solves related but different problem with behavioral approximations.
  - "Reinforcement learning" also solves a behavioral variation (e.g., $\approx$ adaptive expectations)
  - We solve **exact model** with finite # of agents. Will not claim convergence towards continuum.

The curse of dimensionality in equilibrium models

- In any models where the distribution impacts decisions (and vice-versa), agents need to keep track and forecast their own states and the states of everyone else.

- Three components to the curse of dimensionality with many agents (Bellman, 1958, p. IX)

  1. The cardinality of the state space is enormous: memory requirements, update of coefficients, ...
  2. Even with an approximation, you need to evaluate highly-dimensional conditional expectations over every idiosyncratic shock: continuation value function, Euler equations, LOMs,...
  3. Even with both solved, calculating the ergodic distribution/boundary conditions may still be cursed—e.g., fixed point solving for ergodic distribution and agent decisions with iterative algorithm

There is "no free lunch", even with deep learning

- Insights from **symmetry** of problem structure and analogies to "mean-field" limit might help
  - Frequently, distributions can be enough to calculate payoffs (e.g., walrasian auctioneers ignore your name) or "exchangeability" in game theory/IO
  - If there are a lot of agents, the maybe forecasting the distribution might become easier?
- Tradeoff: accept **higher approximation error** for **distant** regions of statespace
  - Dynamics from a (small) finite number of distributional initial conditions. But how given ergodicity/etc.?
- In this paper, we explore these with the classic "investment under uncertainty" model
  - We introduce **permutation-invariant dynamic programming** to formalize symmetry/exchangeability
  - Use this theory to guide a **deep learning** approximation with a simple, non-heuristic algorithm
  - See "Spooky Boundaries" paper for when/how these methods can avoid calculating ergodic distributions

Background: Deep learning for functional equations

Equilibrium models as functional equations

Most theoretical models in economics with equilibrium conditions can be written as functional equations:

- Take some function(s) $f \in \mathcal{F}$ where $f : \mathcal{X} \to \mathcal{Y}$ (e.g. asset price, investment choice, best-response).
- Domain $\mathcal{X}$ could be state (e.g. dividends, capital, opponents state) or time if sequential.
- The "model" is $\ell : \mathcal{F} \times \mathcal{X} \to \mathcal{R}$ (e.g., stack Euler and Bellman residuals, equilibrium FOCs).
- Normalize so that a solution is the "zero" of the residuals at each $x \in \mathcal{X}$.

Then a **solution** is an $f^* \in \mathcal{F}$ where $\ell(f^*, x) = 0$ for all $x \in \mathcal{X}$. How do we find approximate solution?

Interpolation solutions for solving functional equations

Classic approach: use class of functions with finite parameters and interpolate a finite number of points

1. **Pick** finite set of N points $\hat{\mathcal{X}} \subset \mathcal{X}$ (e.g., a grid).
2. **Choose** approximation $\hat{f}(\cdot;\theta) \in \mathcal{H}(\Theta)$ with parameters $\Theta \subseteq \mathbb{R}^M$ (e.g., polynomials, splines).
3. **Fit** with nonlinear least-squares for a general $M \gtreqless N$

$$\min_{\theta \in \Theta} \sum_{x \in \hat{\mathcal{X}}} \ell(\hat{f}(\cdot;\theta), x)^2$$

   - If $\theta \in \Theta$ is such that $\ell(\hat{f}(\cdot;\theta), x) = 0$ for all $x \in \hat{\mathcal{X}}$ we say it **interpolates** $\hat{\mathcal{X}}$.

4. **Verify** that $\hat{f}(x;\theta) \approx f^*(x)$ for $x \in \mathcal{X} \setminus \hat{\mathcal{X}}$. i.e., has low **generalization error**
   - For $M \geq N$ we usually interpolate exactly (and hence $\hat{f}(x;\theta) \approx f^*(x)$ for $x \in \hat{\mathcal{X}}$).

Deep learning here just enables "pick, choose, fit, hope" with more flexibility using **economic insights**.

"Modern" ML is massively overparameterized

Deep learning here is **highly-overparameterized** $\mathcal{H}$ (i.e. $M \gg N$) designed for good generalization:

- Choose $\mathcal{H}$ using economic insights (e.g. encode symmetry) given problem structure from ℓ and $\mathcal{F}$
- Composing $\mathcal{H}$ from multiple functions (e.g., "deep"er) tends to generalize better in practice.
- For example, if $f : \mathbb{R}^Q \to \mathbb{R}$ could choose $\hat{f}(x;\theta) \equiv W_2 \cdot \sigma(W_1 \cdot x + b_1) + b_2$:
  - $\sigma(\cdot) = max(0, \cdot)$ element-wise (i.e. ReLU in CS literature) but many variations.
  - $W_1 \in \mathbb{R}^{P \times Q}$, $b_1 \in \mathbb{R}^P$, $W_2 \in \mathbb{R}^P$, and $b_2 \in \mathbb{R}$, and $\theta \in \Theta \equiv \{b_1, W_1, b_2, W_2\}$
  - Try adding another "layer": $\hat{f}(x;\theta) \equiv W_3 \cdot \sigma(W_2 \cdot \sigma(W_1 \cdot x + b_1) + b_2) + b_3$ or more structure.
- Software (e.g., PyTorch) makes it easy to experiment with different $\mathcal{H}$ (i.e., neural networks), manage the θ, calculate $\nabla_\theta \ell(\hat{f}(\cdot;\theta), x)$ required for optimizers (i.e., auto-differentiation)

Deep learning optimizes in a space of functions

- $\mathcal{H}(\Theta)$ is more general, but the objective hasn't changed i.e. $\min_{\theta \in \Theta} \sum_{x \in \hat{\mathcal{X}}} \ell(\hat{f}(\cdot; \theta), x)^2$.
- Since $M \gg N$, massive multiplicity of θ where $\hat{f}(\cdot; \theta)$ interpolates, and objective value ≈ 0
- Since individual θ are irrelevant it is helpful to think of optimization directly within $\mathcal{H}$

$$\min_{\hat{f} \in \mathcal{H}} \sum_{x \in \hat{\mathcal{X}}} \ell(\hat{f}, x)^2 \tag{1}$$

Which $\hat{f}$?

Deep learning optimization interpolates with an "inductive bias"

- **Counterintuitively:** for M large enough, optimizers **tend to** converge towards something **unique** $\hat{f}$ in equivalence class from some $\|\cdot\|_S$ define on $x \in \mathcal{X}$ (i.e., not just at interpolated "data").
- **Mental model:** chooses min-norm interpolating solution for a (usually) unknown functional norm S

$$\min_{\hat{f} \in \mathcal{H}} ||\hat{f}||_S$$
$$\text{s.t. } \ell(\hat{f}, x) = 0, \quad \text{for } x \in \hat{\mathcal{X}}$$

  - CS literature refers to this as the **inductive bias**: optimization process biased towards particular $\hat{f}$
  - Intuition is that it may choose the interpolating solutions which are flattest and have smallest derivatives.
- Is $\|\hat{f} - f^*\|_S$ small (i.e., does the min-norm solution generalize well)?
  - "No free lunch theorem" in optimization/ML. Depends on $\ell, \mathcal{H}$ and $\hat{\mathcal{X}}$.

In this paper: show how to design $\mathcal{H}$; and implement expectations in ℓ to solve high-dimensional equilibria with finite numbers of agents which generalize well on $\mathcal{X}$ given x_0

Application

Our application

A variation of the Lucas and Prescott (1971) model of investment under uncertainty with N firms.

Why?

1. Ljungqvist and Sargent (2018), pp. 226-228, use it to introduce recursive competitive equilibria.
2. Simple model that fits in one slide.
3. Under one parameterization, the model has a known LQ solution, which gives us an exact benchmark: We can show that our solution will be extremely accurate.
4. By changing one parameter, the model is nonlinear and, yet, our method handles the nonlinear case as easily as the LQ case and, according to the Euler residuals, with high accuracy.

A permutation-invariant economy

- Industry consisting of $N > 1$ firms, each producing the same good.
- A firm i produces output x with x units of capital.
- Thus, the vector $X \equiv [x_1, \dots x_N]^\top$ is the production (or capital) of the whole industry.
- The inverse demand function for the industry is, for some $\nu \geq 1$ (this is our twist!):

$$p(X) = 1 - \frac{1}{N} \sum_{i=1}^{N} x_i^\nu$$

- The firm does not consider the impact of its individual decisions on $p(X)$.
- Due to adjustment frictions, investing u has a cost $\frac{\gamma}{2} u^2$.
- Law of motion for capital $x' = (1 - \delta)x + u + \sigma w + \eta\omega$ where $w \sim \mathcal{N}(0, 1)$ an i.i.d. idiosyncratic shock, and $\omega \sim \mathcal{N}(0, 1)$ an i.i.d. aggregate shock, common to all firms.
- The firm chooses u to maximize $\mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t \left(p(X)x - \frac{\gamma}{2} u^2\right)\right]$.

Recursive problem

The recursive problem of the firm taking the exogenous policy $\hat{u}(\cdot, X)$ for all other firms as given is:

$$
\begin{aligned}
v(x, X) = \max_{u} \left\{ p(X)x - \frac{\gamma}{2}u^2 + \beta \mathbb{E}\left[v(x', X')\right] \right\} \\
\text{s.t. } x' = (1-\delta)x + u + \sigma w + \eta\omega \\
X_i' = (1-\delta)X_i + \hat{u}(X_i, X) + \sigma W_i + \eta\omega, \quad \text{for } i \in \{1, ..., N\}
\end{aligned}
$$

Take FOCs and equation using standard steps to write equilibrium as the LOM and Euler equation

$$\gamma u(x, X) = \beta \mathbb{E}\left[p(X') + \gamma(1-\delta)u(x', X')\right]$$

Goal: Use problem structure to design $\mathcal{H}$ class for $u(x, X)$ approximation

A 'big X, little x' dynamic programming problem

Consider:

$$v(x, X) = \max_u \left\{ r(x, u, X) + \beta \mathbb{E}\left[v(x', X') \right] \right\}$$
$$\text{s.t. } x' = g(x, u) + \sigma w + \eta \omega$$
$$X' = G(X) + \Omega W + \eta \omega \mathbf{1}_N$$

where:

1. x is the individual state of the agent.
2. X is a vector stacking the individual states of all of the N agents in the economy.
3. u is the control.
4. w is random innovation to the individual state, stacked in $W \sim \mathcal{N}(\mathbf{0}_N, \mathbf{I}_N)$ and where, w.l.o.g., $w = W_1$.
5. $\omega \sim \mathcal{N}(0, 1)$ is a random aggregate innovation to all the individual states.

Permutation Groups

- A permutation matrix is a square matrix with a single 1 in each row and column and zeros everywhere else.
- Let $\mathcal{S}_N$ be the set of all $n!$ permutation matrices of size $N \times N$. For example:

$$S_2 = \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right\}$$

- Multiplying vector $v \in \mathbb{R}^N$ by $\pi \in S_N$ reorders elements of v
- (If you know about this): $\mathcal{S}_N$ is the *symmetric group* under matrix multiplication.

Permutation-invariant dynamic programming

A 'big X, little x' dynamic programming problem is a **permutation-invariant dynamic programming problem** if, for all $(x, X) \in \mathbb{R}^{N+1}$ and all permutations $\pi \in \mathcal{S}_N$

1. The reward function r is **permutation invariant**:

$$r(x, u, \pi X) = r(x, u, X)$$

2. The deterministic component of the law of motion for X is **permutation equivariant**:

$$G(\pi X) = \pi G(X)$$

3. The covariance matrix of the idiosyncratic shocks satisfies

$$\pi \Omega = \Omega \pi$$

Permutation invariance of the optimal solution

Proposition

The optimal solution of a permutation-invariant dynamic programming problem is permutation invariant. That is, for all $\pi \in \mathcal{S}_N$*:*

$$u(x, \pi X) = u(x, X)$$

and:

$$v(x, \pi X) = v(x, X)$$

Can $u(x, X)$ permutation invariance guide $\mathcal{H}$ choice?

Curse of dimensionality in this example

Recall there are three separate sources of the "curse" here.

1. Can we approximate $u(x, X)$ for high dimensional $X \in \mathbb{R}^N$ without massive increases in the $\mathcal{X}$ grid?
   - Only if $u(x, X)$ "generalizes" well from limited $\hat{\mathcal{X}}$—otherwise grids on $\hat{\mathcal{X}}$ grow exponentially.
   - Remember goal: fitting $\hat{\mathcal{X}}$ always happens with enough points, but with smart $\mathcal{H}$ choice and "inductive bias" might generalize well for small $\hat{\mathcal{X}}$
2. Given intuition that individual X_i have limited affect on $u(x, X)$, how to calculate $\mathbb{E}\left[u(x', X')\right]$?
   - Look at $\mathbb{E}\left[u(x', X') \mid x', \omega\right]$ to condition on firm's idiosyncratic state x' and aggregate shock ω
   - Maybe the dimensionality $X' \in \mathbb{R}^N$ is a blessing, not a curse?
3. Can we avoid boundary conditions that require ergodic solutions for X evolution?
   - Would focus on limited set of X_0 help if we care more about error on X_5 than X_∞?
   - See "Spooky Boundaries" paper

Main result I: Representation of permutation-invariant functions

Proposition

(based on Wagstaff et al., 2019) Let $f : \mathbb{R}^{N+1} \to \mathbb{R}$ *be a continuous permutation-invariant function under* $\mathcal{S}_N$*, i.e., for all* $(x, X) \in \mathbb{R}^{N+1}$ *and all* $\pi \in \mathcal{S}_N$*:*

$$f(x, \pi X) = f(x, X)$$

Then, there exist a latent dimension $L \leq N$ *and continuous functions* $\rho : \mathbb{R}^{L+1} \to \mathbb{R}$ *and* $\phi : \mathbb{R} \to \mathbb{R}^L$ *such that:*

$$f(x, X) = \rho\left(x, \frac{1}{N}\sum_{i=1}^{N} \phi(X_i)\right)$$

- This proposition should remind you of Krusell-Smith!
- Key benefit for approximation is the **representation** (ρ, ϕ), **not dimensionality reduction**.
- Fitting a ρ and ϕ rather than f directly leads to **far better generalization** on $\mathcal{X}$.
- Faster to fit to $\hat{\mathcal{X}}$ and $L \ll N$ in practice—but generalization is our goal, not interpolation speed.

Main result II: Concentration of measure

Proposition

Concentration of measure when expected gradients are bounded in N Suppose $z \sim \mathcal{N}(\mathbf{0}_N, \Sigma)$, where the spectral radius of Σ, denoted by $\rho(\Sigma)$, is independent of N, z^1 a draw from z, and $f : \mathbb{R}^N \to \mathbb{R}$ is a function with expected gradient bounded in N. Then:

$$\mathbb{P}\left(\left|f(z^1) - \mathbb{E}\left[f(z)\right]\right| \geq \epsilon\right) \leq \frac{\rho(\Sigma)C}{\epsilon^2}\frac{1}{N}$$

- As Ledoux (2001) puts it: "A random variable that depends in a Lipschitz way on many independent variables (but not too much on any of them) is essentially constant."
- With concentration of measure, dimensionality is not a curse; it is a blessing!

Concretely, for our problem can calculate $\mathbb{E}\left[u(x', X') \mid w, \omega\right]$ with a *single draw* of idiosyncratic shocks W

Solving the Model

Our deep learning architectures

- We will specify several deep learning architectures $\mathcal{H}(\theta)$:
  1. ϕ is approximated as a function of a finite set of moments à la Krusell-Smith but in a fully nonlinear way as in Fernández-Villaverde et al. (2019). We use 1 and 4 moments.
  2. ϕ is approximated by a flexible ReLU network with two layers in ϕ and 128 coefficients).
- The baseline ϕ(Identity), ϕ(Moments), and ϕ(ReLU) have 49.4K, 49.8K, and 66.8K coefficients respectively regardless of N.
- In all cases, ρ is a highly parameterized neural network with four layers.
- A surprising benefit of a high-dimensional approximation is the "double-descent" phenomenon in machine learning (see Belkin et al., 2019, and Advani et al., 2020): more coefficients makes it easier to find minimum-norm solutions.
- All the code in `PyTorch` but very easy to implement in any ML framework.

Solution method follows "interpolation" methods

1. **Pick**: $\hat{\mathcal{X}}$ as simulated trajectories from X_0. Only need dozens/hundreds of points $\hat{\mathcal{X}}$ regardless of N
2. **Choose**: implement the $\mathcal{H}$ with u with ρ and ϕ as discussed
3. **Fit**: residual $\varepsilon(X;u) \equiv \gamma u(X) - \beta \mathbb{E}\left[P(X') + \gamma(1-\delta)u(X')\right]$ using LOM for X'

$$\min_{\hat{u}\in\mathcal{H}} \sum_{X\in\hat{\mathcal{X}}} \varepsilon(X;\hat{u})^2$$

- But don't forget the better mental model is that this finds a particular interpolating solution

$$\min_{\hat{u}\in\mathcal{H}} ||\hat{u}||_S$$
$$\text{s.t. } \varepsilon(X;\hat{u}) = 0, \quad \text{for } X \in \hat{\mathcal{X}}$$

4. **Verify**: Norm S unknown, but check $\varepsilon(X;\hat{u})$ sample/simulate by drawing $X \in \mathcal{X} \setminus \hat{\mathcal{X}}$ from X_0

Study two cases: linear ($\nu = 1$) and nonlinear ($\nu > 1$) demand functions

Case 1: Linear to verify algorithms and methods

- With $\nu = 1$, we have a linear demand function: $p(X) = 1 - \frac{1}{N}\sum_{i=1}^{N} x_i$.
- It generates an LQ dynamic programming problem (only the mean of x_i matters!).
- We can find the exact $u(x, X)$ using the linear regulator solution.
- The LQ solution gives us a benchmark against which we can compare our deep learning solution.
- The neural network "learns" very quickly that the solution is $u(x, X) = H_0 + \frac{1}{N} H_1 \sum_{i=1}^{N} x_i$ and finds a high-dimensional approximation which matches that for the training grid.
- We also compute a modified linear regulator solution with *one* Monte Carlo draw instead of setting the individual shocks to zero: illustrates how concentration of measure works.
- Bonus point: we show how to implement this modified linear regulator solution. Useful for non-Gaussian LQ problems where certainty equivalence does not hold.

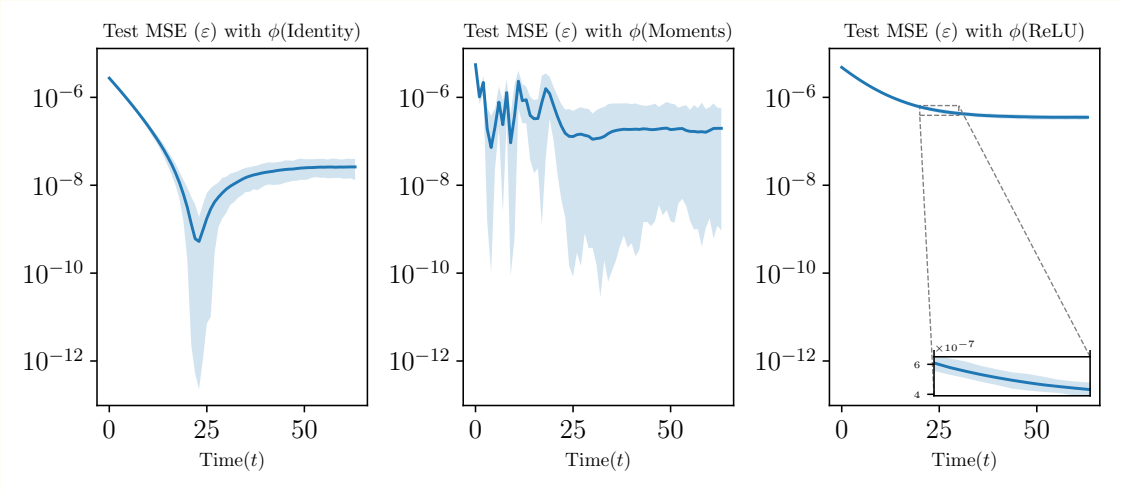


Figure 1: The Euler residuals for $\nu = 1$ and $N = 128$ for ϕ(Identity), ϕ(Moments), and ϕ(ReLU). The dark blue curve shows the average residuals along equilibrium paths for 256 different trajectories. The shaded areas depict the 2.5th and 97.5th percentiles.

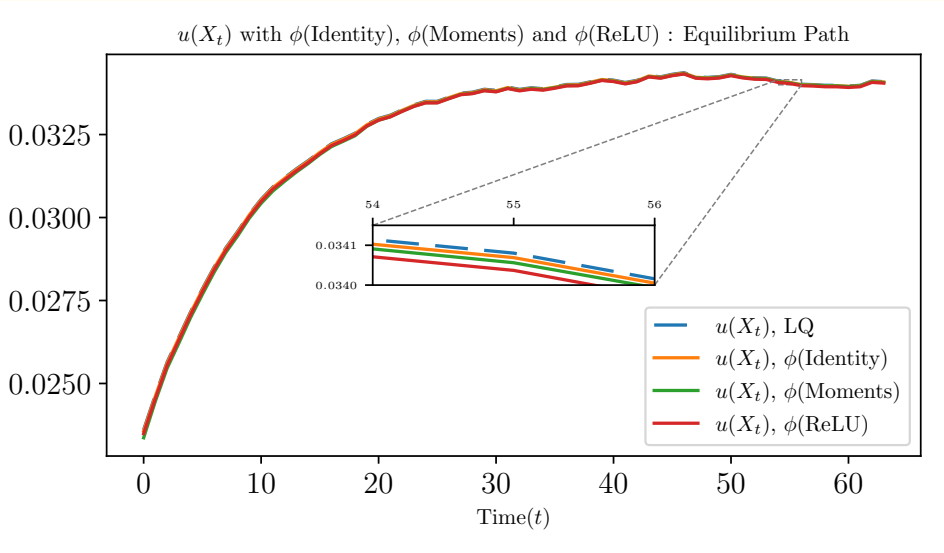


Figure 2: Comparison between baseline approximate solutions and the LQ-regulator solution for the case with $\nu = 1$ and $N = 128$.

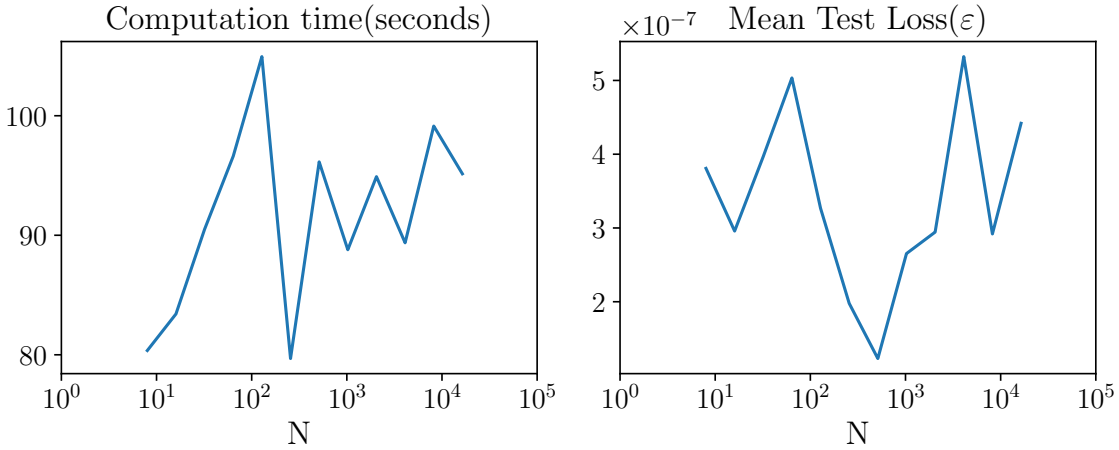


Figure 3: Performance of the ϕ(ReLU) for different N (median value of 21 trials).

Case 2: Nonlinear case with no "closed-form" solution

- With $\nu > 1$, we have a nonlinear demand function: $p(X) = 1 - \frac{1}{N}\sum_{i=1}^{N} x_i^{\nu}$.
- Notice how, now, the whole distribution of of x_i matters!
- But we can still find the solution to this nonlinear case using exactly the same functional approximation and algorithm as before.
- We do not need change anything in the code except the value of ν.
- Since the LQ solution no longer holds, we do not have an exact solution to use as a benchmark, but can check residuals.
- Same model and method. Computation time by N nearly the same to linear case

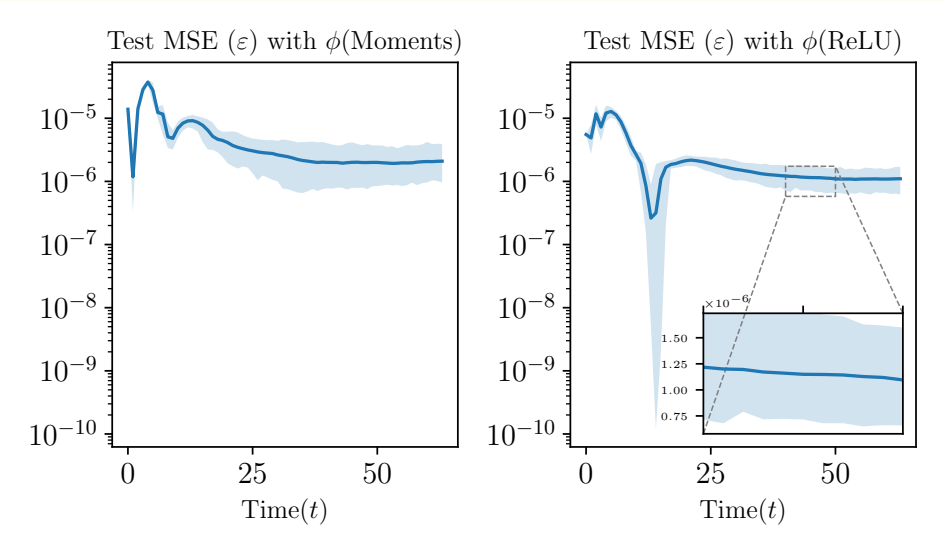


Figure 4: The Euler residuals for $\nu = 1.5$ and $N = 128$ for ϕ(Moments) and ϕ(ReLU). The dark blue curve shows the average residuals along equilibrium paths for 256 different trajectories. The shaded areas depict the 2.5th and 97.5th percentiles.

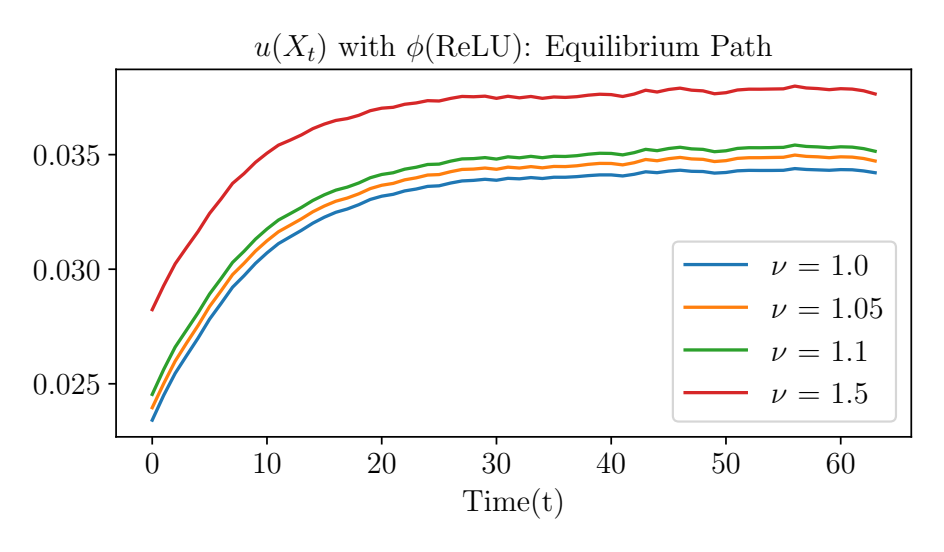


Figure 5: The optimal policy u along the equilibrium paths for $\nu = [1.0, 1.05, 1.1, 1.5]$ and $N = 128$. Each path shows the optimal policy for a single trajectory.

Generalizability and Approximation Error

Representation with linear prices

Recall the representation,

$$u(x, X) = \rho\left(x, \frac{1}{N}\sum_{i=1}^{N}\phi(X_i)\right)$$

Can show that the following **exact solution** holds with our representation

- $\phi(X_n) = X_n$ identity, and $L = 1$
- $\rho(x, y) = \theta_1 + \theta_2 y$
- Doesn't matter how to generate X since only need 2 points

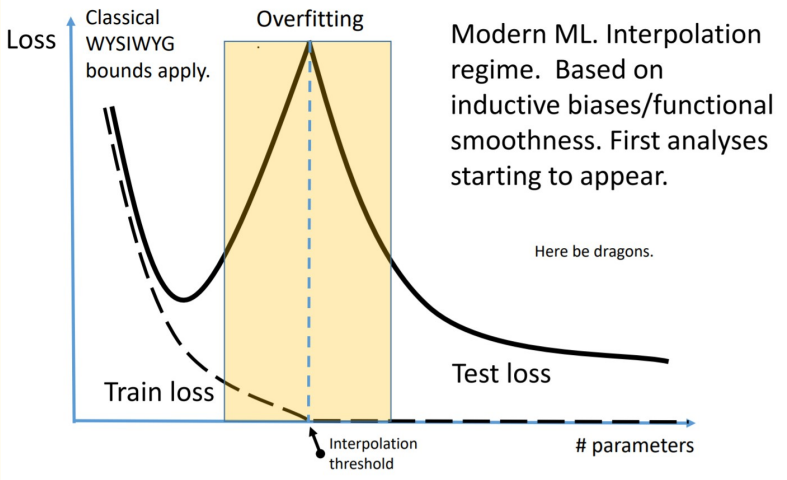
They may let you reflect on symmetry and summary statistics but no surprises so far. But...

Extreme example of generalizability of neural networks

- Forgot we know any closed form, and see if overfitting hurts us.
- Fit **three** data points in $\mathbb{R}^{512}$
- Flexible functional form with 17.7 K parameters
- Now, evaluate for a whole bunch of reasonable trajectories from the initial condition and check the policy error
  - 5×10^{-5} MSE of euler, approximately **0.06%** relative error of $u(X)$

The deep-learning generalization theory to explain this is emerging.

The cure to overfitting is to add more parameters



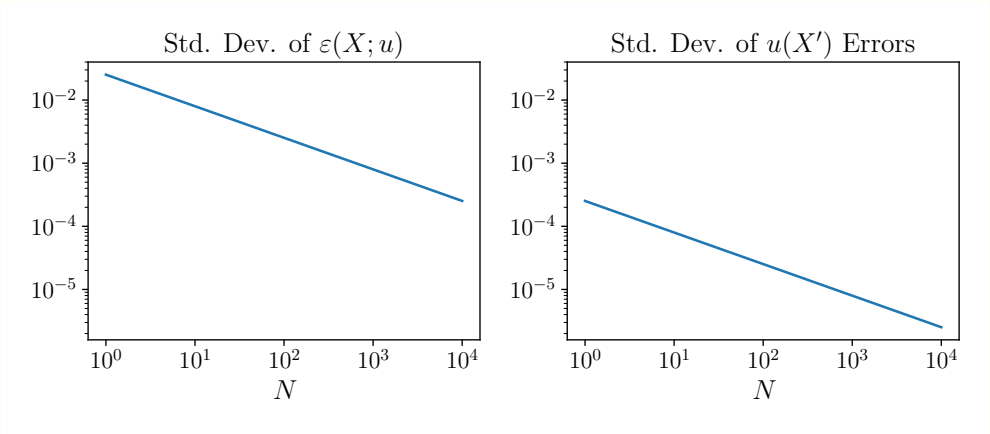
Belkin et al., 2019—traditional statistics/bias-variance tradeoff stop around the interpolation threshold

Concentration of measure is the bless of dimensionality

$$\mathbb{E}\left[P(X') + \gamma u(X') \mid \omega\right] \approx P(\hat{X}') + \gamma u(\hat{X}'), \text{ for draw of } \hat{X}' \mid \omega$$

- Conditional expectation becomes constant as N gets large!
- Can calculate the expectation with a **single Monte-carlo draw**
  - Draw $\hat{X}'$ conditioned on ω since u, P depend "a lot" on it
- Check $P(X)$ doesn't depend too much on any $X_i \in X$
  - e.g. is expected gradient bounded in N
  - $u(\cdot)$ properties can follow from $P(\cdot)$
- Back to "continuum trick"
  - It worked because the $P(\cdot)$ and $u(\cdot)$ don't depend on any one agent outside of x (i.e. not sensitive to measure zero changes)
  - Very robust result, especially easy to fulfill with symmetric functions

Analytic euler error due to the concentration of measure



Euler Error with with one draw $\hat{X}$ using LOM. Recall $\varepsilon \equiv -\gamma u(X) + \beta P(\hat{X}') + \gamma(1-\delta)u(\hat{X}')$

Conclusions

Extensions

1. Decreasing returns to scale: the policy becomes a function of x.
2. Multiple productivity types.
3. Complex idiosyncratic states.
4. Global solutions with transitions and aggregate shocks.
5. Many different network architectures.

Summarizing our contribution

- Primary goal: new tools to solve (previously) **intractable models** (e.g. finite # of agents)
- **Method** for solving **high-dimensional** dynamic programming problems and competitive equilibria
  - Dynamic models with heterogeneity & idiosyncratic/aggregate shocks
  - Global and with transitions, and yet no need for backwards induction or even a calculations of a steady-state
  - Dimensionality is a **bless**; only a curse in low to medium dimensions
- Using **deep learning for function approximation** with a huge # of parameters ($\gg$ grid points)
  - Standard recursive economics. No agent-based modeling. No reinforcement learning
- Reevaluate **Krusell-Smith** and how far it can be pushed with symmetry+deep learning
- Some teasers from the implementation
  - 10000+ dimensional state-spaces are not a problem
  - 10000+ dimensional expectations with one Monte-carlo draw
  - Fit linear case with 17.7 K parameters fit with only 3 data points!

Appendix

Table 1: Performance of Different Networks in Solving the Linear Model

group	description	Time (s)	Params (K)	Train MSE (ε)	Test MSE (ε)	Val MSE (ε)	Policy Error ($\lvert u - u_{\text{ref}} \rvert$)	Policy Error $\left(\frac{\lvert u - u_{\text{ref}} \rvert}{u_{\text{ref}}}\right)$
ϕ(Identity)	Baseline	42	49.4	4.1e-06	3.3e-07	3.3e-07	2.9e-05	0.10%
	Thin (64 nodes)	33	12.4	3.7e-06	2.7e-07	2.7e-07	3.4e-05	0.10%
ϕ(Moments)	Baseline	55	49.8	1.4e-06	7.6e-07	7.6e-07	2.8e-05	0.09%
	Moments (1,2)	211	49.5	2.4e-06	1.1e-06	2.3e-06	4.4e-05	0.14%
	Very Shallow(1 layer)	241	0.6	1.1e-05	8.4e-06	7.9e-06	1.1e-02	34.00%
	Thin (64 nodes)	82	12.6	1.6e-06	9.1e-07	9.2e-07	3.8e-05	0.12%
ϕ(ReLU)	Baseline	107	66.8	3.7e-06	3.3e-07	3.3e-07	2.7e-05	0.09%
	L = 2	86	66.3	1.3e-05	2.1e-07	2.2e-07	2.6e-05	0.08%
	L = 16	91	69.9	5.5e-06	1.5e-07	1.5e-07	2.1e-05	0.07%
	Shallow(ϕ : 1 layer, ρ : 2 layers)	79	17.7	2.0e-06	5.5e-07	5.5e-07	3.2e-05	0.11%
	Deep(ϕ : 4 layers, ρ : 8 layers)	242	165.1	2.1e-03	2.2e-03	2.1e-03	2.7e-03	8.50%
	Thin(ϕ, ρ : 64 nodes)	87	17.0	1.1e-05	4.5e-07	4.5e-07	3.0e-05	0.10%

Table 2: Nonlinear Model Performance

group	description	Time (s)	Params (K)	Train MSE (ε)	Test MSE (ε)	Val MSE (ε)
ϕ(Moments)	Baseline	26	49.8	6.0e-06	5.0e-06	3.8e-06
	Moments (1)	24	49.4	2.7e-05	6.5e-06	3.4e-06
	Moments (1,2)	27	49.5	8.0e-06	5.1e-06	3.6e-06
	Very Shallow (1 layer)	252	0.6	8.3e-06	1.4e+00	5.0e-06
	Shallow (2 layers)	35	17.0	5.8e-06	1.2e+00	4.4e-06
	Thin (32 nodes)	66	3.2	1.1e-05	9.7e-06	4.4e-06
ϕ(ReLU)	Baseline	60	67.1	1.4e-05	4.7e-06	3.3e-06
	L = 1	109	66.3	9.4e-06	1.3e-05	4.5e-06
	L = 2	73	66.6	1.0e-05	3.3e-06	2.3e-06
	L = 8	73	68.1	1.1e-05	4.9e-06	2.0e-06
	L = 16	72	70.2	1.5e-05	5.4e-06	1.7e-06
	Very Shallow(ϕ, ρ : 1 layer)	136	1.4	8.9e-06	4.8e+06	4.9e-06
	Shallow(ϕ, ρ : 2 layers)	47	34.3	1.0e-05	9.2e-06	2.8e-06
	Thin(ϕ, ρ : 32 nodes)	52	4.5	1.3e-05	6.0e-06	2.7e-06

Comparing Performance: More Different Network Designs (Linear)

group	description	Time (s)	Params (K)	Train MSE (ε)	Test MSE (ε)	Val MSE (ε)	Policy Error ($\lvert u - u_{\text{ref}} \rvert$)	Policy Error $\left(\frac{\lvert u - u_{\text{ref}} \rvert}{u_{\text{ref}}}\right)$
ϕ(Identity)	Baseline	42	49.4	4.1e-06	3.3e-07	3.3e-07	2.9e-05	0.10%
	Thin (64 nodes)	33	12.4	3.7e-06	2.7e-07	2.7e-07	3.4e-05	0.10%
	Shallow (2 layers)	159	16.6	3.7e-06	7.8e-07	7.6e-07	9.4e-03	33.53%
ϕ(Moments)	Baseline	55	49.8	1.4e-06	7.6e-07	7.6e-07	2.8e-05	0.09%
	Moments (1,2)	211	49.5	2.4e-06	1.1e-06	2.3e-06	4.4e-05	0.14%
	Very Shallow(1 layer)	241	0.6	1.1e-05	8.4e-06	7.9e-06	1.1e-02	34.00%
	Shallow (2 layers)	137	17.0	1.6e-06	9.9e-07	9.5e-07	1.8e-02	59.41%
	Deep(8 layers)	241	115.3	2.8e-06	1.2e-06	1.0e-06	5.2e-05	0.16%
	Thin (64 nodes)	82	12.6	1.6e-06	9.1e-07	9.2e-07	3.8e-05	0.12%
	Wide (256 nodes)	61	197.9	1.8e-06	8.7e-07	8.0e-07	4.3e-05	0.13%
ϕ(ReLU)	Baseline	107	66.8	3.7e-06	3.3e-07	3.3e-07	2.7e-05	0.09%
	L = 1	88	66.0	1.8e-05	2.3e-07	2.4e-07	2.8e-05	0.10%
	L = 2	86	66.3	1.3e-05	2.1e-07	2.2e-07	2.6e-05	0.08%
	L = 8	70	67.8	3.0e-05	5.9e-07	5.9e-07	3.3e-05	0.11%
	L = 16	91	69.9	5.5e-06	1.5e-07	1.5e-07	2.1e-05	0.07%
	Shallow(ϕ : 1 layer, ρ : 2 layers)	79	17.7	2.0e-06	5.5e-07	5.5e-07	3.2e-05	0.11%
	Shallow(ϕ : 1 layer)	58	50.4	8.7e-06	1.5e-07	1.5e-07	2.5e-05	0.08%
	Shallow(ρ : 2 layers)	89	34.0	3.1e-06	4.2e-07	4.2e-07	2.7e-05	0.09%

Comparing Performance: Different Networks Designs (nonlinear)

group	description	Time (s)	Params (K)	Train MSE (ε)	Test MSE (ε)	Val MSE (ε)
ϕ(Moments)	Baseline	26	49.8	6.0e-06	5.0e-06	3.8e-06
	Moments (1)	24	49.4	2.7e-05	6.5e-06	3.4e-06
	Moments (1,2)	27	49.5	8.0e-06	5.1e-06	3.6e-06
	Very Shallow (1 layer)	252	0.6	8.3e-06	1.4e+00	5.0e-06
	Shallow (2 layers)	35	17.0	5.8e-06	1.2e+00	4.4e-06
	Thin (32 nodes)	66	3.2	1.1e-05	9.7e-06	4.4e-06
ϕ(ReLU)	Baseline	60	67.1	1.4e-05	4.7e-06	3.3e-06
	L $= 1$	109	66.3	9.4e-06	1.3e-05	4.5e-06
	L $= 2$	73	66.6	1.0e-05	3.3e-06	2.3e-06
	L $= 8$	73	68.1	1.1e-05	4.9e-06	2.0e-06
	L $= 16$	72	70.2	1.5e-05	5.4e-06	1.7e-06
	Very Shallow(ϕ, ρ : 1 layer)	136	1.4	8.9e-06	4.8e+06	4.9e-06
	Shallow(ϕ, ρ : 2 layers)	47	34.3	1.0e-05	9.2e-06	2.8e-06