Solving Heterogeneous Agents Models with DeepHAM and Reinforcement Learning

Yucheng Yang

University of Zurich and SFI

based on work with Jiequn Han & Weinan E

Turino Summer School on Deep Learning for Economics and Finance

August 26, 2025

Major Development in Macroeconomics

"Major focus of macro over the last 20 years has been the development of models that incorporate rich specifications of heterogeneity and frictions that can simultaneously can speak to aggregate outcomes while also addressing a rich set of cross-sectional facts."

- Richard Rogerson (2021).

Some Prominent Examples:

- Uneven dynamics across sectors and population after major economic and policy shocks.
- Aggregate impact of monetary/fiscal policy through heterogeneous households & firms.
- Distribution of investors matter for asset prices.

Paradigm shift: Representative agent models $\rightarrow$ Heterogeneous agent models.

New challenge: distribution and its feedback (to decision and aggregates).

Introduction

- Heterogeneous agent (HA) models with aggregate shocks are solved with global Krusell-Smith (KS) method or local perturbation method.

	KS method	Perturbation method
Multiple shocks	No	Yes
Multiple endogenous states	No	Yes
Estimation/Calibration	No	Yes
Large shocks	Yes	No
Risky steady state	Yes	No
Nonlinearity e.g. ZLB	Yes	No

HA Model with Aggregate Shocks: Krusell and Smith (1998) Model

- Production economy with a continuum of households: each household i solves

$$\max_{c_{i,t}\geq 0, a_{i,t+1}\geq \underline{a}} \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t u\left(c_{i,t}\right)$$

  subject to budget constraint

$$a_{i,t+1} = w_t y_{i,t} + R_t a_{i,t} - c_{i,t}$$

- Idiosyncratic shocks on employment status $y_{i,t} \in \{y_b, y_g\}$.
- Representative firm produces $Y_t = Z_t F(K_t, \bar{L})$.
- Aggregate shock $Z_t \in \{Z_b, Z_g\}$.
- Markov chain that describes the joint evolution of the exogenous shocks:

$$\pi\left(z', y' \mid z, y\right) = \Pr\left(z_{t+1} = z', y_{t+1} = y' \mid z_t = z, y_t = y\right)$$

Krusell and Smith (1998) Model (Ct'd)

- Production side: representative firm produces $Y_t = Z_t F(K_t, \bar{L})$.
- Perfect competition + optimization $\Rightarrow$ factor prices:

$$R_t = Z_t \partial_K F(K_t, \bar{L}) - \delta, \qquad w_t = Z_t \partial_L F(K_t, \bar{L})$$

- Capital and labor market clearing $\Rightarrow K_t = \int a_{i,t} di$, $\bar{L} = \int y_{i,t} di$.
- Factor prices are uncertain because of aggregate shock.

Recursive Competitive Equilibrium

- Recursive form of household i's problem: DeepHAM

$$V(a_i, y_i, Z, \mathbf{\Gamma}) = \max_{c_i, a_i'} \{u(c_i) + \beta \mathbb{E} V\left(a_i', y_i', Z', \mathbf{\Gamma}' | y_i, Z\right)\}$$

  subject to budget and borrowing constraints $c + a' = w(Z, K(\Gamma))y + R(Z, K(\Gamma))a, a' \geq 0$, and perceived law of motion $\Gamma' = \Psi\left(Z, \Gamma, Z'\right)$.

- $\mathbf{\Gamma}$: distribution over (a, y) of all households.

- Households must know $\mathbf{\Gamma}$ to predict factor prices $\Rightarrow$ infinite dimensional $\mathbf{\Gamma}$ is state variable. Why?

- Euler equation with saving policy $a' = g(a, y, Z, \Gamma)$:

$$u_c(Ra + wy - g(a, y, Z, \Gamma)) \geq$$
$$\beta \mathbb{E}\left[R\left(Z', \Gamma'\right) u_c\left(R\left(Z', \Gamma'\right) g(a, y, Z, \Gamma) + w\left(Z', \Gamma'\right) y' - g\left(g(a, y, Z, \Gamma), y', Z', \Gamma'\right)\right)\right]$$

- To solve for g, households need to forecast prices next period, which depend on Γ'

- Agents need to know the equilibrium law of motion Ψ to forecast Γ' given Γ.

Computational Setup: Krusell-Smith Method

- Curse of dimensionality shows up in recursive form of household i's problem: DeepHAM

$$V(a_i, y_i, Z, \mathbf{\Gamma}) = \max_{c_i, a_i'} \{u(c_i) + \beta \mathbb{E} V\left(a_i', y_i', Z', \mathbf{\Gamma}' | y_i, Z\right)\}$$

  subject to budget and borrowing constraints. $\mathbf{\Gamma}$: distribution over (a, y) of all households.

- Krusell-Smith method (KS, 1998; Maliar et al., 2010):

  1. Approximate state: $\hat{s}_i = (a_i, y_i, Z, m_1)$. m_1: first moment of individual asset distribution.
  2. Perceived law of motion (PLM) for m_1: log linear form

$$\log(m_{1,t+1}) = A(Z) + B(Z) \log(m_{1t}).$$

Computational Setup: Krusell-Smith Method

- Curse of dimensionality shows up in recursive form of household i's problem: DeepHAM

$$V(a_i, y_i, Z, \mathbf{\Gamma}) = \max_{c_i, a_i'} \{u(c_i) + \beta \mathbb{E} V\left(a_i', y_i', Z', \mathbf{\Gamma}' | y_i, Z\right)\}$$

  subject to budget and borrowing constraints. $\mathbf{\Gamma}$: distribution over (a, y) of all households.

- Krusell-Smith method (KS, 1998; Maliar et al., 2010):

  1. Approximate state: $\hat{s}_i = (a_i, y_i, Z, m_1)$. m_1: first moment of individual asset distribution.
  2. Perceived law of motion (PLM) for m_1: log linear form

$$\log(m_{1,t+1}) = A(Z) + B(Z)\log(m_{1t}).$$

- Very costly in complex HA models with multiple assets or multiple shocks.
- Question: Is it solving the original problem?

Computational Setup: Krusell-Smith Method

- Curse of dimensionality shows up in recursive form of household i's problem: DeepHAM

$$V(a_i, y_i, Z, \mathbf{\Gamma}) = \max_{c_i, a_i'} \{u(c_i) + \beta \mathbb{E} V\left(a_i', y_i', Z', \mathbf{\Gamma}' | y_i, Z\right)\}$$

  subject to budget and borrowing constraints. $\mathbf{\Gamma}$: distribution over (a, y) of all households.

- Krusell-Smith method (KS, 1998; Maliar et al., 2010):

  1. Approximate state: $\hat{s}_i = (a_i, y_i, Z, m_1)$. m_1: first moment of individual asset distribution.
  2. Perceived law of motion (PLM) for m_1: log linear form

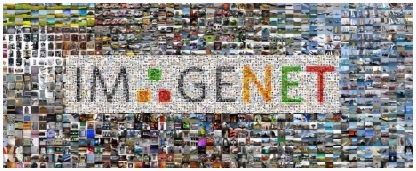
$$\log(m_{1,t+1}) = A(Z) + B(Z)\log(m_{1t}).$$

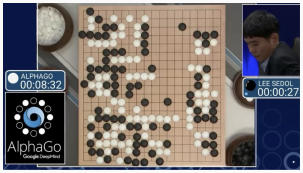
- Very costly in complex HA models with multiple assets or multiple shocks.
- Question: Is it solving the original problem? No!
- New approach: global solution method using deep learning.

Deep Learning for High Dimensional Models

In recent years, deep learning has achieved remarkable success in many areas, including:

- High dimensional prediction problem (classification or regression).
- High dimensional policy learning.
- Generative models.





Deep learning has also led to path-breaking work in high dimensional scientific problems in many disciplines.

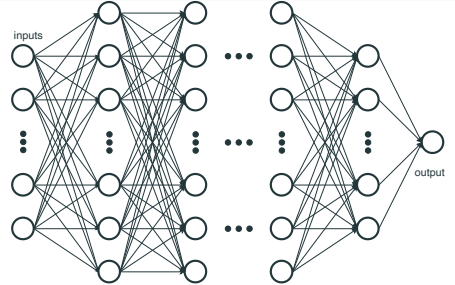
Deep Neural Networks: Formulation

- Representation: in a compositional form rather than additive,

$$f(x) = \mathcal{L}^P \circ \cdots \circ \mathcal{L}^p \circ \cdots \circ \mathcal{L}^1(\mathbf{x}), \quad \mathbf{x}\text{: high-dim state vector},$$

$$\mathbf{h}_p = \mathcal{L}^p(\mathbf{h}_{p-1}) = \sigma\big(\mathbf{W}_p\mathbf{h}_{p-1} + \mathbf{b}_p\big), \quad \mathbf{h}_0 = \mathbf{x},$$

- $\sigma(x)$: element-wise nonlinear fn: e.g., $\max(0, x)$, Tanh$(\cdot)$, sigmoid $\frac{1}{1+e^{-x}}$, etc.
- Want to solve unknown parameters $\boldsymbol{\Theta} = \{\mathbf{W}_p, \mathbf{b}_p\}_p$.



Deep Learning: Optimization and Implementation

Key steps to "learn" high-dim functions:

- Empirical objective function (chosen by economists):

$$\min_{\theta} \mathcal{L}(x_{1:N}; \theta) = \min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(f(x_i; \theta)).$$

- Algorithm: stochastic gradient descent (SGD) and its variants.
  - Gradient descent: $\theta \leftarrow \theta - \eta \nabla \mathcal{L}(x_{1:N}, \theta)$.
  - SGD: $\theta \leftarrow \theta - \eta \nabla \mathcal{L}(x_i; \theta)$; Mini-batch SGD: $\theta \leftarrow \theta - \eta \nabla \mathcal{L}(x_{i:i+n}; \theta)$.
  - Actually used: Adam (Adaptive Moment Estimation) optimizer.
- Gradients are efficiently calculated with backpropagation (chain rule) in deep learning packages like TensorFlow, PyTorch, and JAX.

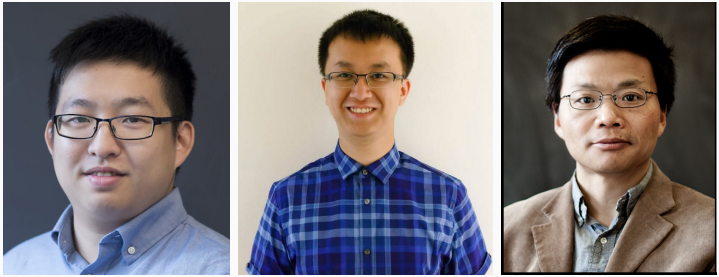
Similar procedure, but more efficient than polynomial approximation: strong representation capability, and computational efficiency.

DeepHAM method

DeepHAM: A Global Solution Method for Heterogeneous Agent Models with Aggregate Shocks

Jiequn Han[*], Yucheng Yang[†], and Weinan E[‡]

This version: January 2025
First version: December 2021



DeepHAM: Represent Distribution with Neural Networks (NN)

- Consider N-agent Krusell-Smith problem (N finite but large). General form of value & policy functions are like (ignore y): Krusell-Smith setup

$$V(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z), \quad c(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z)$$

DeepHAM: Represent Distribution with Neural Networks (NN)

- Consider N-agent Krusell-Smith problem (N finite but large). General form of value & policy functions are like (ignore y): Krusell-Smith setup

$$V(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z), \quad c(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z)$$

- Encode economics in NN architecture: approximate distn with permutation invariant generalized moments $\frac{1}{N}\sum_i \mathcal{Q}(a_i)$, basis function $\mathcal{Q}(\cdot)$ parameterized by (sub) NN:

$$\widetilde{V}(a_i; \frac{1}{N}\sum_i \mathcal{Q}_1(a_i), \ldots, \frac{1}{N}\sum_i \mathcal{Q}_J(a_i); Z)$$

$$\widetilde{c}(a_i; \frac{1}{N}\sum_i \tilde{\mathcal{Q}}_1(a_i), \ldots, \frac{1}{N}\sum_i \tilde{\mathcal{Q}}_J(a_i); Z)$$

DeepHAM: Represent Distribution with Neural Networks (NN)

- Consider N-agent Krusell-Smith problem (N finite but large). General form of value & policy functions are like (ignore y): Krusell-Smith setup

$$V(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z), \quad c(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z)$$

- Encode economics in NN architecture: approximate distn with permutation invariant generalized moments $\frac{1}{N}\sum_i \mathcal{Q}(a_i)$, basis function $\mathcal{Q}(\cdot)$ parameterized by (sub) NN:

$$\widetilde{V}(a_i; \frac{1}{N}\sum_i \mathcal{Q}_1(a_i), \ldots, \frac{1}{N}\sum_i \mathcal{Q}_J(a_i); Z)$$

$$\widetilde{c}(a_i; \frac{1}{N}\sum_i \tilde{\mathcal{Q}}_1(a_i), \ldots, \frac{1}{N}\sum_i \tilde{\mathcal{Q}}_J(a_i); Z)$$

- Special case: $\mathcal{Q}(a) = a$ yields the first moment.

DeepHAM: Represent Distribution with Neural Networks (NN)

- Consider N-agent Krusell-Smith problem (N finite but large). General form of value & policy functions are like (ignore y): Krusell-Smith setup

$$V(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z), \quad c(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z)$$

- Encode economics in NN architecture: approximate distn with permutation invariant generalized moments $\frac{1}{N}\sum_i \mathcal{Q}(a_i)$, basis function $\mathcal{Q}(\cdot)$ parameterized by (sub) NN:

$$\widetilde{V}(a_i; \frac{1}{N}\sum_i \mathcal{Q}_1(a_i), \ldots, \frac{1}{N}\sum_i \mathcal{Q}_J(a_i); Z)$$

$$\widetilde{c}(a_i; \frac{1}{N}\sum_i \tilde{\mathcal{Q}}_1(a_i), \ldots, \frac{1}{N}\sum_i \tilde{\mathcal{Q}}_J(a_i); Z)$$

- Special case: $\mathcal{Q}(a) = a$ yields the first moment.
- Algorithm solves generalized moments (GMs) that matter most for policy and value functions. ("numerically determined sufficient statistics")

DeepHAM: Represent Distribution with Neural Networks (NN)

- Consider N-agent Krusell-Smith problem (N finite but large). General form of value & policy functions are like (ignore y): Krusell-Smith setup

$$V(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z), \quad c(a_i; a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_N; Z)$$

- Encode economics in NN architecture: approximate distn with permutation invariant generalized moments $\frac{1}{N}\sum_i \mathcal{Q}(a_i)$, basis function $\mathcal{Q}(\cdot)$ parameterized by (sub) NN:

$$\widetilde{V}(a_i; \frac{1}{N}\sum_i \mathcal{Q}_1(a_i), \ldots, \frac{1}{N}\sum_i \mathcal{Q}_J(a_i); Z)$$

$$\widetilde{c}(a_i; \frac{1}{N}\sum_i \tilde{\mathcal{Q}}_1(a_i), \ldots, \frac{1}{N}\sum_i \tilde{\mathcal{Q}}_J(a_i); Z)$$

- Special case: $\mathcal{Q}(a) = a$ yields the first moment.
- Algorithm solves generalized moments (GMs) that matter most for policy and value functions. ("numerically determined sufficient statistics")
- GMs provide interpretabilitiy on how heterogeneity matters.

DeepHAM Algorithm: General Procedure

- Formulate discrete time N-agent HA models, solve value and policy functions parameterized by neural nets $V(s), c(s)$. $s = (a_i, y_i, Z, \mathbf{\Gamma})$.
- Parameterize two parts of mapping:
  1. Distribution $\mathbf{\Gamma} \mapsto J$ generalized moments $\frac{1}{N}\sum_i \mathcal{Q}_j(a_i)$.
  2. $(a_i, y_i, Z, \{\frac{1}{N}\sum_i \mathcal{Q}_j(a_i)\}) \mapsto c, V$.

DeepHAM Algorithm: General Procedure

- Formulate discrete time N-agent HA models, solve value and policy functions parameterized by neural nets $V(s), c(s)$. $s = (a_i, y_i, Z, \mathbf{\Gamma})$.
- Parameterize two parts of mapping:
  1. Distribution $\mathbf{\Gamma} \mapsto J$ generalized moments $\frac{1}{N}\sum_i \mathcal{Q}_j(a_i)$.
  2. $(a_i, y_i, Z, \{\frac{1}{N}\sum_i \mathcal{Q}_j(a_i)\}) \mapsto c, V$.
- Iteratively update value and policy functions. In each iteration:
  1. Simulate stationary distribution with the latest policy.
  2. Given policy function, update value function with supervised learning (regression).
  3. Given value function, optimize policy function on simulated paths.

DeepHAM: Value Function Learning

Define cumulative utility for HH i up to t:

$$\widetilde{U}_{i,t} = \sum_{\tau=0}^{t} \beta^{\tau} u\left(c_{i,\tau}\right).$$

In iteration k, given policy function $\mathcal{C}^{(k-1)}(s)$:

1. Sample states s from the stationary distribution. Then the value of each state s can be approximately calculated as cumulative utility in the following T (T large enough) periods following policy $\mathcal{C}^{(k-1)}(s)$:

$$\widetilde{V}^{(k)}(s) \approx \mathbb{E}\widetilde{U}_T = \mathbb{E}\sum_{\tau=0}^{T} \beta^{\tau} u\left(c_{i,\tau}\right)$$

2. Learn value function $V^{(k)}(s)$ parameterized by deep neural networks with regression/supervised learning. Function input = states, output = truncated value. Sample: states in the simulation.

DeepHAM: Policy Function Optimization

In iteration k, given $V^{(k)}(s)$, optimize policy $\mathcal{C}^{(k)}(s)$ on simulated paths.

Fictitious play: In N-agent competitive equilibrium problem, when solving agent i's problem, fix other agents' policy from last "play". Iterate the following:

1. At "play" $\ell+1$, last play's policy $\mathcal{C}^{(k,\ell)}(s)$ is known.
2. For agent $i=1$, update her optimal policy $\mathcal{C}^{(k,\ell+1)}(s)$ according to:

$$\max_{\mathcal{C}^{(k,\ell+1)}(s)} \mathbb{E}_{\mu(\mathcal{C}^{(k-1)}),\mathcal{E}} \left(\sum_{t=0}^{T} \beta^t u\left(c_{i,t}\right) + \beta^T V^{(k)}(s_{i,T}) \right)$$

   subject to others all following $\mathcal{C}^{(k,\ell)}(s)$ in the first T periods.
3. All agents adopt the new policy $\mathcal{C}^{(k,\ell+1)}(s)$ in "play" $\ell+1$.

DeepHAM: Policy Function Optimization

In iteration k, given $V^{(k)}(s)$, optimize policy $\mathcal{C}^{(k)}(s)$ on simulated paths.

Fictitious play: In N-agent competitive equilibrium problem, when solving agent i's problem, fix other agents' policy from last "play". Iterate the following:

1. At "play" $\ell+1$, last play's policy $\mathcal{C}^{(k,\ell)}(s)$ is known.
2. For agent $i=1$, update her optimal policy $\mathcal{C}^{(k,\ell+1)}(s)$ according to:

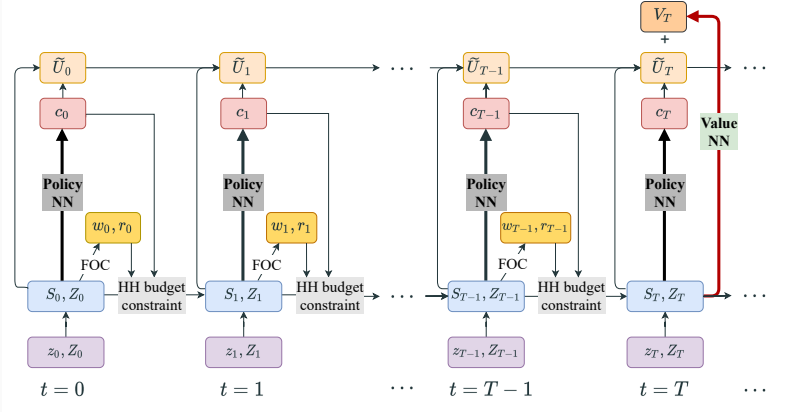
$$\max_{\mathcal{C}^{(k,\ell+1)}(s)} \mathbb{E}_{\mu(\mathcal{C}^{(k-1)}),\mathcal{E}} \left(\sum_{t=0}^{T} \beta^t u\left(c_{i,t}\right) + \beta^T V^{(k)}(s_{i,T}) \right)$$

   subject to others all following $\mathcal{C}^{(k,\ell)}(s)$ in the first T periods.
3. All agents adopt the new policy $\mathcal{C}^{(k,\ell+1)}(s)$ in "play" $\ell+1$.

Optimization solved on Monte Carlo simulation with N agents on a large number of sample paths in a computational graph.

Computational Graph for Policy Function Optimization

$$\max_{\Theta^C} \mathbb{E}_{\mu(\mathcal{C}^{(k-1)}),\mathcal{E}} \left(\widetilde{U}_{i,T} + \beta^T V_{\mathsf{NN}}(s_{i,T}; \Theta^V) \right)$$



Budget constraint $a_{i,t+1} = (r_t + 1 - \delta)a_{i,t} + w_t \bar{\ell} y_{i,t} - c_{i,t}$. $s_t = (a_{i,t}, y_{i,t}, Z_t, \mathbf{\Gamma}_t)$. Cumulative utility $\widetilde{U}_{i,t} = \sum_{\tau=0}^{t} \beta^\tau u\left(c_{i,\tau}\right)$

Remarks on optimization over simulated paths

- Agents formulate expectation over future via long simulation: no perceived law of motion needed.
  - Similar to the idea of reinforcement learning.
  - Not rely on equilibrium conditions: solve models that are difficult to derive FOCs (e.g. non-convex problem, planner's problem).
  - Can be extended to study problems beyond full information rational expectation equilibrium (ongoing work with Ben Moll, Chiyuan Wang, and others)

Remarks on optimization over simulated paths

- Agents formulate expectation over future via long simulation: no perceived law of motion needed.
  - Similar to the idea of reinforcement learning.
  - Not rely on equilibrium conditions: solve models that are difficult to derive FOCs (e.g. non-convex problem, planner's problem).
  - Can be extended to study problems beyond full information rational expectation equilibrium (ongoing work with Ben Moll, Chiyuan Wang, and others)
- Our formulation: easily extend to constrained efficiency (planner's) problem.
  - Competitive equilibrium: fictitious play.
  - Constrained efficiency: optimize all agents' policy together.

Remarks on optimization over simulated paths

- Agents formulate expectation over future via long simulation: no perceived law of motion needed.
  - Similar to the idea of reinforcement learning.
  - Not rely on equilibrium conditions: solve models that are difficult to derive FOCs (e.g. non-convex problem, planner's problem).
  - Can be extended to study problems beyond full information rational expectation equilibrium (ongoing work with Ben Moll, Chiyuan Wang, and others)
- Our formulation: easily extend to constrained efficiency (planner's) problem.
  - Competitive equilibrium: fictitious play.
  - Constrained efficiency: optimize all agents' policy together.
- Finite agent approximation + fictitious play: can be used to solve strategic equilibrium.

Implementation Code of DeepHAM

- Our (official) version in **TensorFlow**: `https://github.com/frankhan91/DeepHAM`
- **PyTorch** version by Marko Irisarri (University of Manchester): `https://github.com/markoirisarri/UnofficialDeepHAMPytorchImplementation`
- **JAX** version by Chiyuan Wang (Peking University): `https://github.com/leafDancer/DeepHAMX`

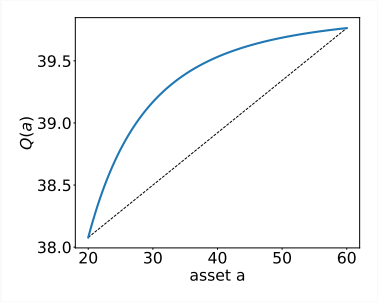
Accuracy Results for Krusell-Smith Problem

Method and Moment Choice	Bellman error	Std of error
KS Method (Maliar et al., 2010)	**0.0253**	0.0002
DeepHAM with 1st moment	0.0184	0.0023
DeepHAM with 1 generalized moments	0.0151	0.0015

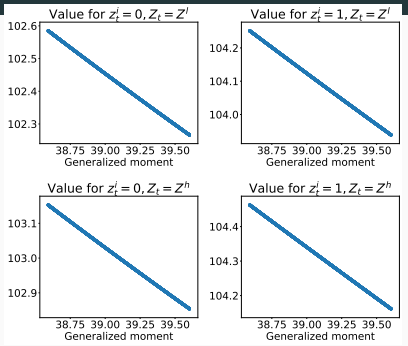
Definition of Bellman Error

- Highly accurate compared to Krusell-Smith (KS) method. solution comparison
- Even only with first moment as model input, DeepHAM outperform KS method due to better capture of nonlinearity.
- Generalized moment yields more accurate solution than the first moment, as it extract more relevant information.

Interpretation of the Generalized Moment (GM)



(a) Plot of $\mathcal{Q}_1(a)$



(b) Map $\frac{1}{N}\sum_i \mathcal{Q}_1(a_i)$ to value fn

- Basis function concave in asset, value function is linear wrt the GM.
- Heterogeneity matters! Unanticipated redistributive policy shock: asset from rich to poor HH $\Rightarrow$ generalized moment$\uparrow\Rightarrow$ unshocked agent welfare$\downarrow$.
- KS method implies no effect, as first moment not change.

DeepHAM for Constrained Efficiency Problem

- Constrained efficiency problem: planner's allocation in incomplete market.
- Important "second best" allocation, but hard to solve in HA models.
- Literature only solves for HA models without aggregate shocks (Davila, Hong, Krusell, Rios-Rull, 2012; Nuno and Moll, 2018).

DeepHAM for Constrained Efficiency Problem

- Constrained efficiency problem: planner's allocation in incomplete market.
- Important "second best" allocation, but hard to solve in HA models.
- Literature only solves for HA models without aggregate shocks (Davila, Hong, Krusell, Rios-Rull, 2012; Nuno and Moll, 2018).
- DeepHAM solves constrained efficiency problem as easily as solve competitive equilibrium, just to remove the fictitious play procedure.
- We solve constrained efficiency problem of Davila et al. (2012), and that with aggregate shocks and countercyclical unemployment risk.
- It takes DeepHAM 20 minutes to solve Davila et al. (2012) on GPU, which takes conventional methods > 10 hours on CPU.

Constrained Efficiency for HA Models w or w/o Agg Shock

	No aggregate shock		Aggregate shock	
	Market	Constrained Opt.	Market	Constrained Opt.
Average assets	30.635	119.741	34.296	95.811
Wealth Gini	0.864	0.862	0.812	0.878
Consumption Gini	0.615	0.386	0.578	0.388

Findings:

1. Both models: with utilitarian planner, K in constrained optimum $\gg$ competitive eqm.
   - Why? Overcome pecuniary externality: $K \uparrow \Rightarrow w \uparrow, R \downarrow$, redistribute from rich to poor (high labor share).

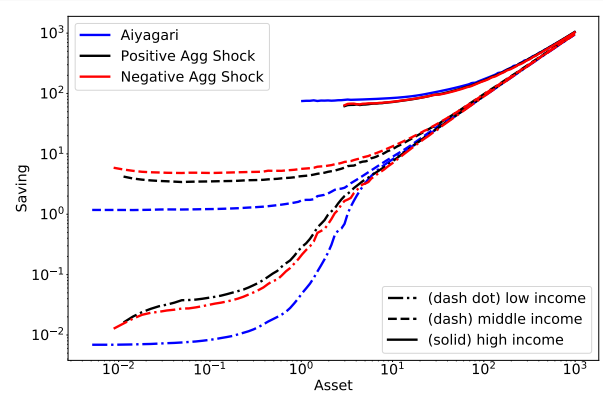
Constrained Efficiency for HA Models w or w/o Agg Shock

	No aggregate shock		Aggregate shock	
	Market	Constrained Opt.	Market	Constrained Opt.
Average assets	30.635	119.741	34.296	95.811
Wealth Gini	0.864	0.862	0.812	0.878
Consumption Gini	0.615	0.386	0.578	0.388

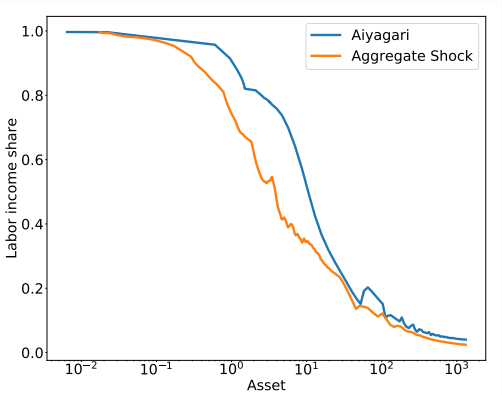
Findings:

1. Both models: with utilitarian planner, K in constrained optimum $\gg$ competitive eqm.
   - Why? Overcome pecuniary externality: $K \uparrow \Rightarrow w \uparrow, R \downarrow$, redistribute from rich to poor (high labor share).
2. Constrained optimal K in model w/ agg shock $<$ w/o agg shock.

Constrained Efficiency for HA Models w or w/o Agg Shock



(c) Policy function

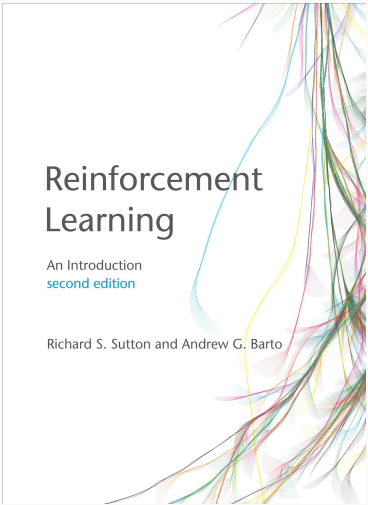


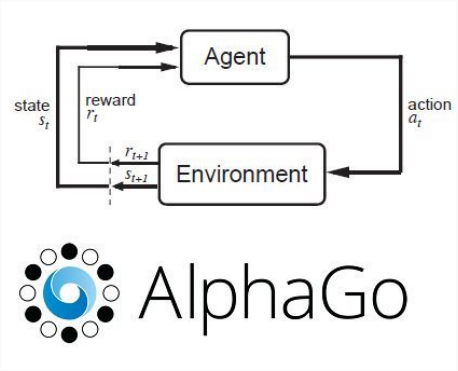
(d) Labor share distribution

Agg shock $\Rightarrow$ precautionary saving $\uparrow$ by poor HHs $\Rightarrow$ labor share lower than model w/o agg shock. So planner raises K less in constrained efficient equilibrium.

More on Reinforcement Learning (RL)

“Optimal control of incompletely-known Markov decision processes” (Sutton-Barto)





Summary of RL Methods (Murphy, 2025)

Approach	Method	Functions learned	On-policy/Off-policy
Value-based	SARSA	$Q(s,a)$	On
Value-based	Q-learning	$Q(s,a)$	Off
Policy-based	REINFORCE	$\pi(a\|s)$	On
Policy-based	A2C	$\pi(a\|s)$, $V(s)$	On
Policy-based	TRPO/PPO	$\pi(a\|s)$, $A(s,a)$	On
Policy-based	DDPG	$a=\pi(s)$, $Q(s,a)$	Off
Policy-based	Soft Actor-Critic	$\pi(a\|s)$, $Q(s,a)$	Off
Model-based	MBRL	$p(s'\|s,a)$	Off

RL Example: TD Learning

A simple problem: given policy function $\pi(x)$, solve for value function.

$$v_\pi(x) = R(x, \pi(x)) + \beta \mathbb{E}_\pi \left[v_\pi \left(x_{t+1} \right) \mid x_t = x \right].$$

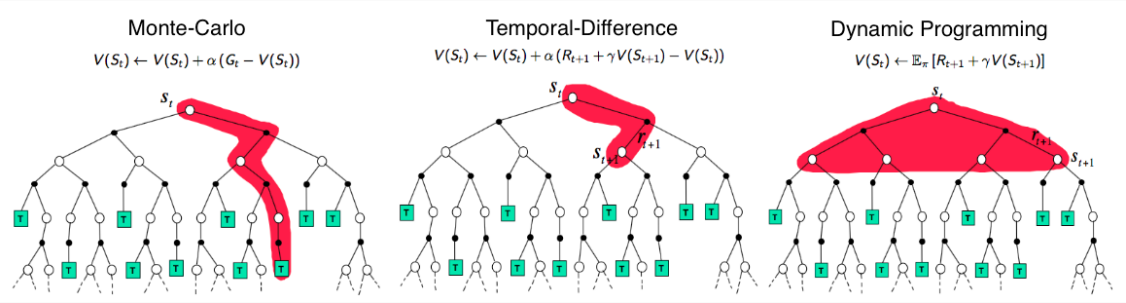
One-step Temporal-difference learning, aka TD(0) (can be generalized to n-step TD(λ)):

- Input: policy $\pi(x)$, step-size $\alpha > 0$, # of time steps T (say $T = 1000$).
- Initial guess $V(x)$ for all $x \in \mathcal{X}$.
- Starting from a randomly drawn initial x_0, for each time step $t = 0, 1, \ldots, T$:
  1. Given the current state x_t and policy π, sample the next state x_{t+1} using the transition probabilities $P\left(\cdot \mid x_t, \pi\left(x_t \right) \right)$
  2. Value prediction: update the guess for the value function according to

$$V\left(x_t \right) \leftarrow V\left(x_t \right) + \alpha [\underbrace{R_t + \beta V\left(x_{t+1} \right)}_{\text{Target}} - V\left(x_t \right)].$$

RL vs Conventional Dynamic Programming

Comparison: Value function learning with RL (Monte Carlo Estimation, TD learning) vs Dynamic Programming (Figure source: Sutton-Barto)



Key: solve value functions with stochastic approximation. Similar logic for policy functions.

The Agenda of "AI for Economics": First Generation

- First generation: "proof of concept" of using AI to solve economic models.
  - Duarte (2018), Azinovic, Gaegauf, Scheidegger (2019), Maliar, Maliar, Winant (2019), Fernández-Villaverde, Hurtado, Nuno (2019), Han, Yang, E (2021), Kahou, Fernández-Villaverde, Perla, Sood (2021), Valaitis and Villa (2021), Gopalakrishna (2022), Huang (2022), Gu, Lauriere, Merkel, Payne (2023), Zhong (2023), among others.
  - All papers solve similar **classical models** in this generation.
- Second generation: apply AI to long-standing difficult questions in various literature. (We'll see more tomorrow and during the conference)

Summary

- We develop DeepHAM, an efficient, reliable, and interpretable deep learning based method to solve HA models with aggregate shocks globally.
- Deep learning based model reduction informs interpretable generalized moments of distribution that matter.
- Reinforcement learning type of objective function and evaluation: allow for solving constrained planner's problem, non-rational expectation equilibrium, etc.
- Brunnermeier et al (2021, RFS): "Neural network techniques will open up a new research avenue in macro-finance."
- As is happening in other disciplines (Nobel Prize Physics 2024, Chemistry 2024), deep learning may become (part of) the new generation of model solvers in economics.

DeepHAM Tutorial Session

DeepHAM Tutorial

Let's go through some key elements of the DeepHAM method in the code.

- General procedure: iteratively update dataset, value function, and policy function.
- Permutation invariant representation of distribution via the generalized moments.
- Update value function with supervised learning.
- Update policy function on the simulated path (computational graph).
- Use fictitious play for competitive equilibrium, or update everyone's policy together for the planner's problem.

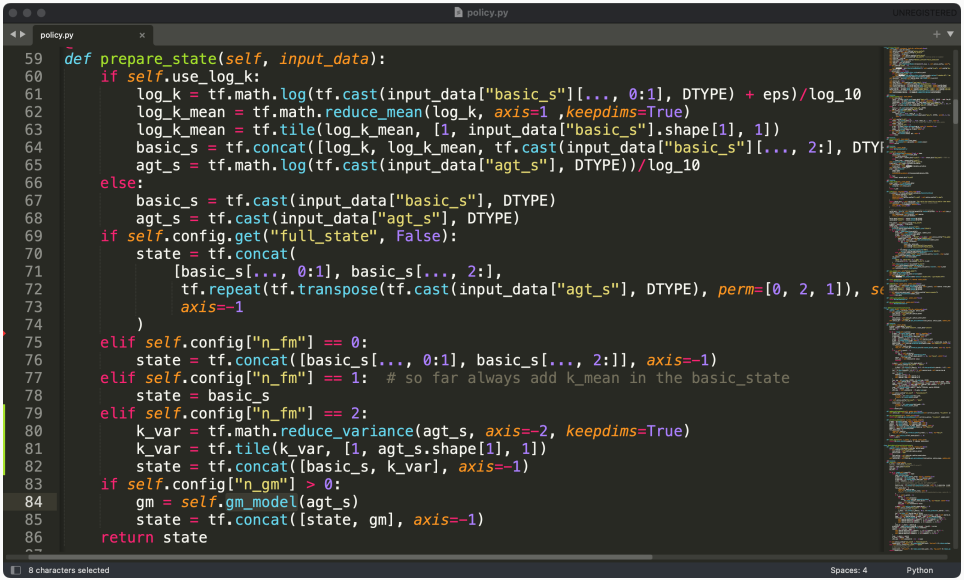
Link to the code: `https://github.com/sischei/SummerSchool_DL_Turin_2025/tree/main/day2/Yang/code/DeepHAM_nuvolos/src`

General procedure: data preparation, value function, and policy function

```python
# initial value training
init_ds = KSInitDataSet(mparam, config)
value_config = config["value_config"]
if config["init_with_bchmk"]:
    init_policy = init_ds.k_policy_bchmk
    policy_type = "pde"
else:
    init_policy = init_ds.c_policy_const_share
    policy_type = "nn_share"
train_vds, valid_vds = init_ds.get_valuedataset(
    init_policy, policy_type, update_init=False
)
vtrainers = []
for i in range(value_config["num_vnet"]):
    config["vnet_idx"] = str(i)
    vtrainers.append(ValueTrainer(config))
for vtr in vtrainers:
    vtr.train(
        train_vds, valid_vds,
        value_config["num_epoch"], value_config["batch_size"]
    )

# iterative policy and value training
policy_config = config["policy_config"]
ptrainer = KSPolicyTrainer(vtrainers, init_ds)
ptrainer.train(policy_config["num_step"], policy_config["batch_size"])
```

Generalized moments



Generalized moments (Ct'd)

util.py

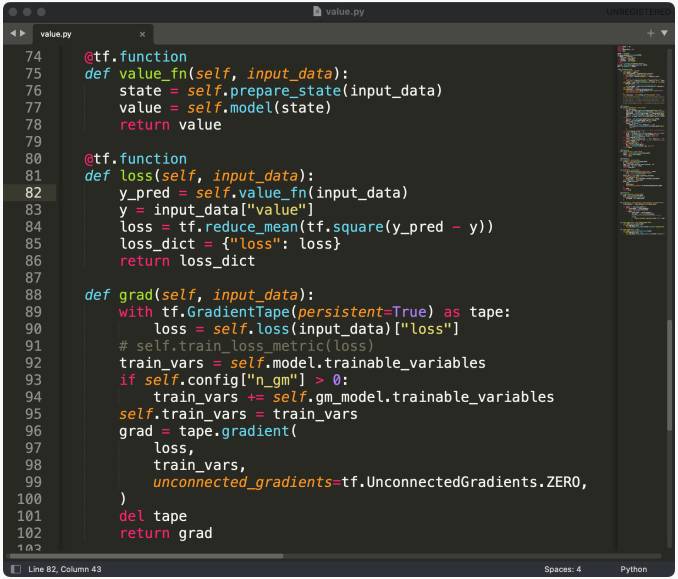
```
class GeneralizedMomModel(FeedforwardModel):
    def __init__(self, d_in, d_out, config, name="generalizedmomentmodel", **kwargs):
        super(GeneralizedMomModel, self).__init__(d_in, d_out, config, name=name, **kwargs)

    def basis_fn(self, inputs):
        x = self.dense_layers[0](inputs)
        for l in self.dense_layers[1:]:
            x = l(x)
        return x

    def call(self, inputs):
        x = self.basis_fn(inputs)
        gm = tf.reduce_mean(x, axis=-2, keepdims=True)
        gm = tf.tile(gm, [1, inputs.shape[-2], 1])
        return gm
```

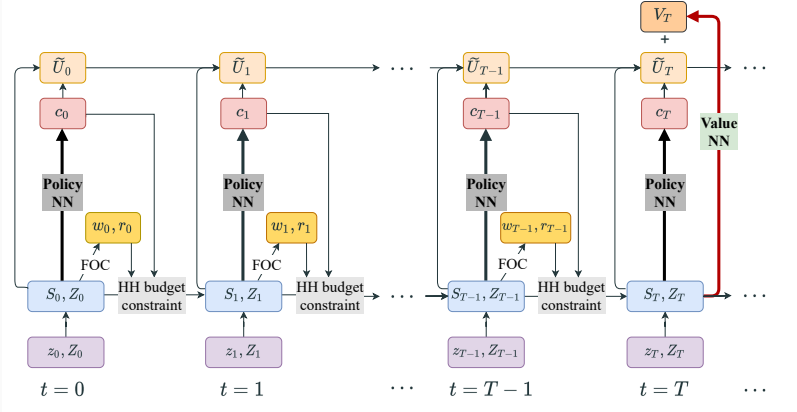
Line 35, Column 1 Spaces: 4

Value Function Updating



Recall: Computational Graph for Policy Function Optimization

$$\max_{\Theta^C} \mathbb{E}_{\mu(\mathcal{C}^{(k-1)}),\mathcal{E}} \left(\widetilde{U}_{i,T} + \beta^T V_{\mathsf{NN}}(s_{i,T}; \Theta^V) \right)$$



Budget constraint $a_{i,t+1} = (r_t + 1 - \delta)a_{i,t} + w_t \bar{\ell} y_{i,t} - c_{i,t}$. $s_t = (a_{i,t}, y_{i,t}, Z_t, \mathbf{\Gamma}_t)$. Cumulative utility $\widetilde{U}_{i,t} = \sum_{\tau=0}^{t} \beta^\tau u\left(c_{i,\tau}\right)$

Policy Function Updating: Cumulative Utility and Fictitious Play

policy.py

```
247         c_share = self.policy_fn(full_state_dict)[..., 0]
248         if self.policy_config["opt_type"] == "game":
249             # optimizing agent 0 only
250             c_share = tf.concat([c_share[:, 0:1], tf.stop_gradient(c_share[:, 1:])], axis=1)
251         # labor tax rate - depend on ashock
252         tau = tf.where(ashock[:, t:t+1] < 1, self.mparam.tau_b, self.mparam.tau_g)
253         # total labor supply - depend on ashock
254         emp = tf.where(
255             ashock[:, t:t+1] < 1,
256             self.mparam.l_bar*self.mparam.er_b,
257             self.mparam.l_bar*self.mparam.er_g
258         )
259         tau, emp = tf.cast(tau, DTYPE), tf.cast(emp, DTYPE)
260         R = 1 - self.mparam.delta + ashock[:, t:t+1] * self.mparam.alpha*(k_mean / emp)**(self.mparam.alpha-1)
261         wage = ashock[:, t:t+1]*(1-self.mparam.alpha)*(k_mean / emp)**(self.mparam.alpha)
262         wealth = R * k_cross + (1-tau)*wage*self.mparam.l_bar*ishock[:, :, t] + \
263             self.mparam.mu*wage*(1-ishock[:, :, t])
264         csmp = tf.clip_by_value(c_share * wealth, EPSILON, wealth-EPSILON)
265         k_cross = wealth - csmp
266         util_sum += self.discount[t] * tf.math.log(csmp)
267
268     if self.policy_config["opt_type"] == "socialplanner":
269         output_dict = {
270             "m_util": -tf.reduce_mean(util_sum),
271             "k_end": tf.reduce_mean(k_cross)
272             }
273     elif self.policy_config["opt_type"] == "game":
274         # optimizing agent 0 only
275         output_dict = {
276             "m_util": -tf.reduce_mean(util_sum[:, 0]),
```

.* Aa “” stop_gradient Find Find Prev

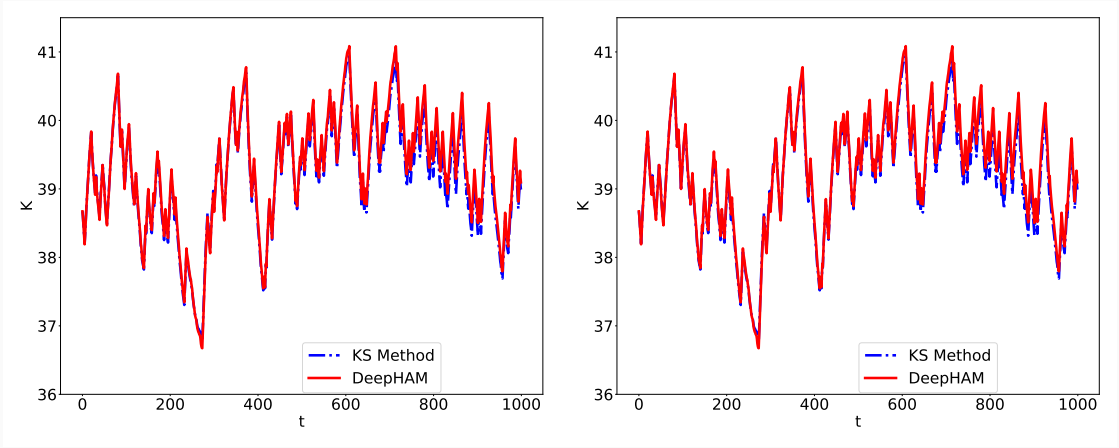
Thank You!

Comments and questions are welcome!

My email: yucheng.yang@uzh.ch

Appendix

Solution Comparison



(e) aggregate capital (K_t)

(f) aggregate consumption (C_t)

back

Accuracy Measures: Bellman Equation Errors

For the KS problem, only using solved value function $V(\cdot)$, Bellman equation error is

$$\text{err}_{\text{B}} = V(a_i, y_i, Z, \mathbf{a}^{-i}, \mathbf{y}^{-i}) - \max_{c_i} \left\{ u(c_i) + \beta \sum_{y', Z', \mathbf{y}'^{-i}} V(a_i', y_i', Z', \widehat{\mathbf{a}'}^{-i}, \mathbf{y}'^{-i}) \times \Pr\left(Z', y'^{i}, \mathbf{y}'^{-i} | Z, y^{i}, \mathbf{y}^{-i}\right) \right\}$$

back