UNIL | Université de Lausanne

Deep Equilibrium Nets*

Simon Scheidegger (UNIL)

Joint work with Marlon Azinović (UZH & SFI) Luca Gaegauf (UZH)

***Current draft available here: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3393482**

Codes here: https://github.com/sischei/DeepEquilibriumNets

Outline

1. **Motivation – why are we doing what we are doing**

2. **Technical background of “Deep Equilibrium Nets”**

   2. I. A simple* benchmark OLG model (due to limited time).

   2. II. From artificial neural networks to “Deep Equilibrium Nets”.

   2. III. Putting things together: A projection method for solving dynamic models.

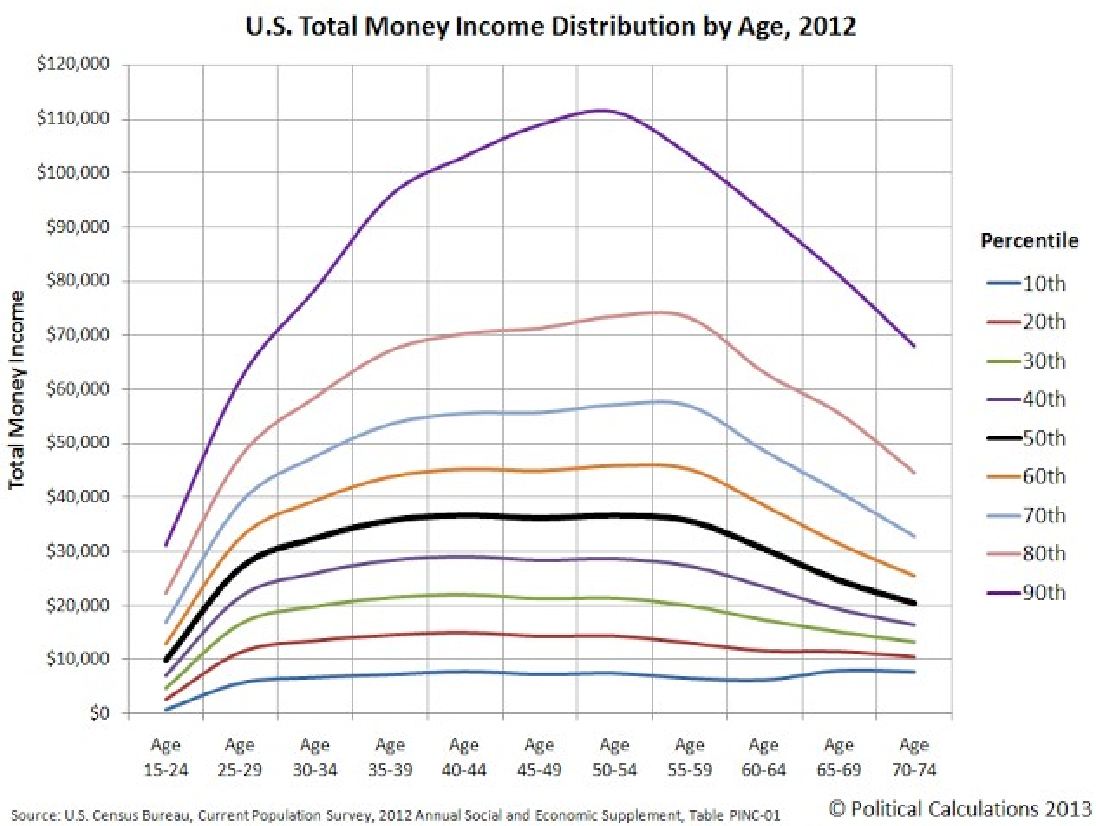
*more comples models (e.g., with liquid/illuqid assets) in paper

Humans are all different – heterogeneous

- **Heterogeneity** a crucial ingredient in contemporary models:
  - to study e.g. cross-sectional consumption response to aggregate shocks.
  - to model, e.g., social security.

- Example OLG models:
  → How many age groups?
  → borrowing constraints?
  → aggregate shocks?
  → idiosyncratic shocks?
  → liquid / illiquid assets**?

→ Models: heterogeneous & high-dimensional



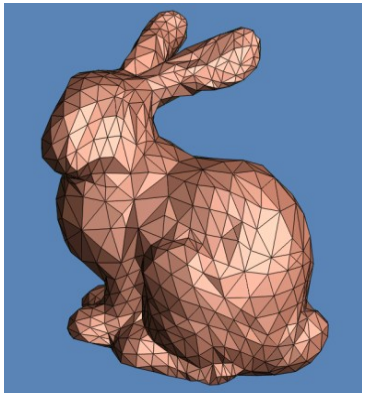
**see, e.g., Kaplan et al. (2018), Wong (2018),...

Dynamic Stochastic Models

e.g. Judd (1998), Ljungquist & Sargent (2004),...

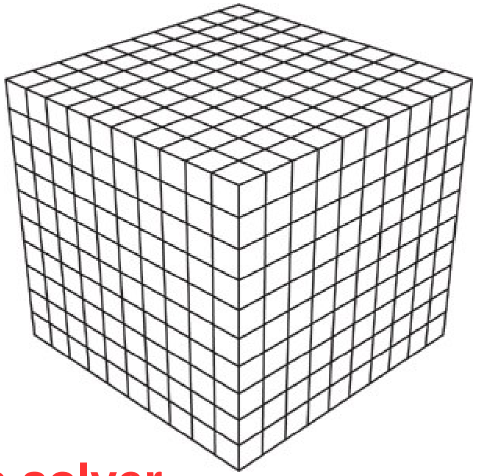
$$\mathbb{E}\left[E\left(\mathbf{x}_t, \mathbf{x}_{t+1}, p\left(\mathbf{x}_t\right), p\left(\mathbf{x}_{t+1}\right)\right) \middle| \mathbf{x}_t, p\left(\mathbf{x}_t\right)\right] = 0$$

$$\mathbf{x}_{t+1} \sim \mathcal{P}\left(\cdot \middle| \mathbf{x}_t, p\left(\mathbf{x}_t\right)\right)$$



x: point in state space; describes your system.

State-space potentially **irregularly-shaped** and **high-dimensional.**



p: time-invariant policy function.

“old solution”:
high-dimensional functions on which **we interpolate**.

→ **N^d** points in ordinary discretization schemes.

→ ***“Curse of dimensionality”.***

→ Need to solve many **non-linear systems of equations** by **invoking a solver**.

What is high-dimensional?

#State Variables (Dimensions)	#Points	Time-to-solution
1	10	10 sec
2	100	~ 1.6 min
3	1,000	~ 16 min
4	10,000	~ 2.7 hours
5	100,000	~ 1.1 days
6	1,000,000	~ 1.6 weeks
...	...	...
20	1e20	3 trillion years (240x age of the universe)

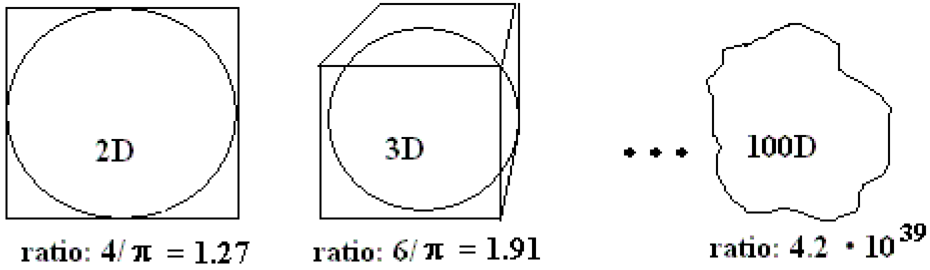
Dimension reduction
Exploit symmetries, e.g., via the active subspace method

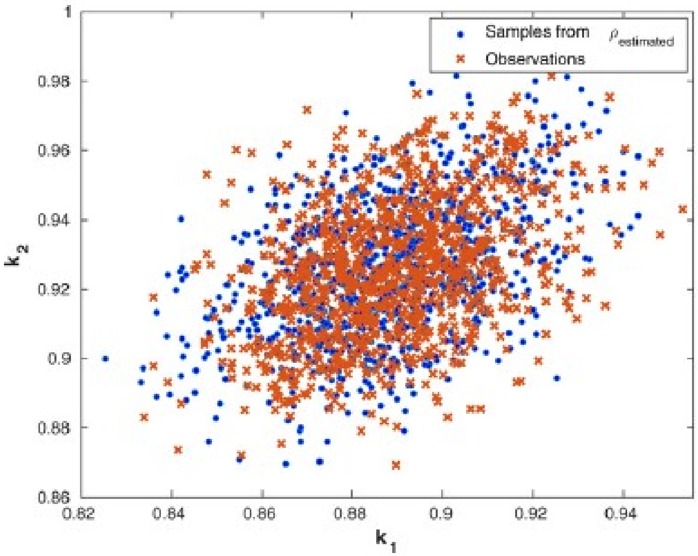
Deal with #Points
e.g., via (Smolyak/adpative) sparse grids

High-performance computing
Reduces time to solution, but not the problem size

Volumes in high dimensions

- Consider a cube of unit lengths containing a sphere of unit radius in higher dimensions.
- For large dimensions: ratio **Volume(Sphere)/Volume(Cube) → 0**





Scheidegger & Bilionis (2019)

Abstract Problem Formulation

i) Contemporary dynamic models: heterogeneous & high-dimensional

ii) Want to solve dynamic stochastic models with high-dimensional state spaces

→ Have to **approximate** and **interpolate** high-dimensional functions on irregular-shaped geometries

→ Problem: curse of dimensionality

iii) **Want to alleviate the curse of dimensionality**

iv) **Want locality of approximation scheme**

v) **Speed-up** → potentially access contemporary HPC systems

Our solution: Deep Equilibrium Nets

→ Solving, e.g., rich OLG models numerically is a **formidable task**.

→ Models are often formulated in a **stylized** fashion to remain computationally **tractable**.

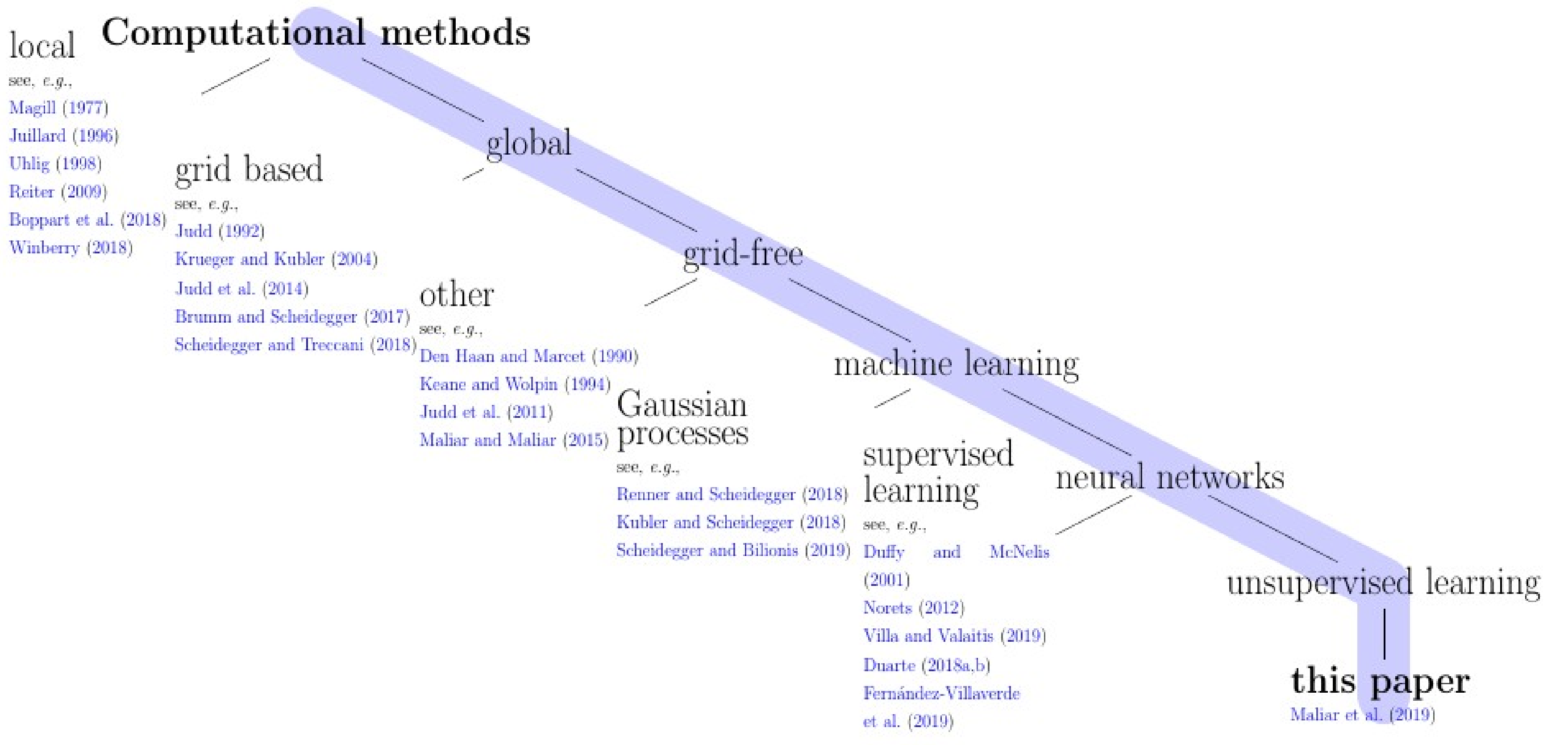
→ We develop a **generic solution framework** based on **neural networks** to solve highly-complex dynamic stochastic models.

Key ideas:

1. Use the **the implied error in the optimality conditions**, as **loss function**.
2. Learn the equilibrium functions with stochastic gradient descent.
3. Take the (training) data points from a simulated path
   → can be generated at **virtual zero cost**.

2. Technical Background

An incomplete list of related literature



2.I. A benchmark OLG model

- Time is discrete: $t = 0, \ldots, \infty$

- Agents live for ***N*** periods (N=60 years).

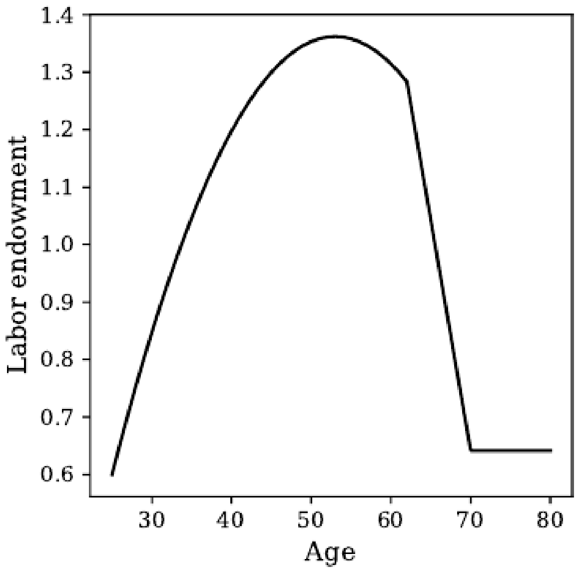
- One representative household per cohort.

- Every *t*, a representative household in born.

- **No uncertainty** about lifetime.

- There are exogenous aggregate shocks ***z*** that follow a Markov chain.

- Each period, the agents alive receive a strictly positive labour endowment which depends on the age of the agent alone.



Households

- Household supplies its labour endowment inelastically for a market wage w_t.

- Agents (with current age s) alive maximize their remaining time-separable discounted expected lifetime utility ($\beta < 1$):

$$\sum_{i=0}^{N-s} \mathrm{E}_t \left[\beta^i u\left(c_{t+i}^{s+i}\right)\right] \quad (1)$$

- Households can save a unit of consumption good to obtain a unit of capital good next period (denoted as a_t^s).

- The savings will become capital in the next period:

$$a_t^s = k_{t+1}^{s+1}, \forall t, \forall s \in \{1, \ldots, N-1\} \quad (2)$$

Households (II)

- Households cannot die with debt.

- Borrowing is allowed up to an exogenously given level: $a_t^s \geq \underline{a}$. (3)

- At time *t*, the households sell their capital to the firm at market price $\mathbf{r}_t > 0$.

- The budget constraint of the household ***s*** in period *t* is

$$c_t^s + a_t^s = r_t k_t^s + l_t^s w_t \tag{4}$$

- The agents are born, and die without any assets $k_t^1 = 0 \text{ and } a_t^N = 0$

Firms & Markets

- There is a single representative firm with Cobb-Douglas production.

- The total factor productivity **η** (**TFP**) and the depreciation **δ** depend on the exogenous shock *z* alone ($\eta(z) \in \{0.85, 1.15\}$, $\delta(z) \in \{0.5, 0.9\}$)

$$\pi^{\delta} = \begin{bmatrix} 0.98 & 0.02 \\ 0.25 & 0.75 \end{bmatrix}, \qquad \pi^{\eta} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \qquad z^{\delta} \otimes z^{\eta} = z \in \{0, 1, 2, 3\}$$

- Each period, after the shock has realized, the firm buys capital and hires labour to maximize its profits, taking prices as given.

- The stochastic production function is given by

$$f(K, L, z) = \eta(z) K^{\alpha} L^{1-\alpha} + K(1 - \delta(z))$$

- There are competitive spot markets for consumption, capital, labor.

Equilibrium

Definition 1 (competitive equilibrium) *A competitive equilibrium, given initial conditions* $z_0, \{k_0^s\}_{s=1}^{N-1}$, *is a collection of choices for households* $\{(c_t^s, a_t^s)_{s=1}^N\}_{t=0}^\infty$ *and for the representative firm* $(K_t, L_t)_{t=0}^\infty$ *as well as prices* $(r_t, w_t)_{t=0}^\infty$, *such that*

1. *Given* $(r_t, w_t)_{t=0}^\infty$, *the choices* $\{(c_t^s, a_t^s)_{s=1}^N\}_{t=0}^\infty$ *maximize* (1), *subject to* (2), (3), *and* (4).
2. *Given* r_t, w_t, *the firm maximizes profits*, i.e.,

$$(K_t, L_t) \in \underset{K_t, L_t \geq 0}{\arg\max} f(K_t, L_t, z_t) - r_t K_t - w_t L_t.$$

3. *All markets clear: For all t*

$$L_t = \sum_{s=1}^N l_t^s,$$

$$K_t = \sum_{s=1}^N k_t^s,$$

(1): max. remaining lifetime utility
(2): savings → capital in next period
(3): borrowing constraint.
(4): budget constraint.

Equilibrium Conditions

- The first order conditions of the firms maximization problem imply

$$w\left(z^t\right) = (1-\alpha)\eta\left(z_t\right) K\left(z^t\right)^{\alpha} L\left(z^t\right)^{-\alpha}$$
$$r\left(z^t\right) = \alpha\eta\left(z_t\right) K\left(z^t\right)^{\alpha-1} L\left(z^t\right)^{1-\alpha} + \left(1-\delta\left(z_t\right)\right)$$

- Optimality conditions for any given generation of age s ∊ 1, ..., N − 1:

$$u'\left(c^s\left(z^t\right)\right) = \beta \mathrm{E}_{z_t}\left[u'\left(c^{s+1}\left(z^t, z_{t+1}\right)\right) r\left(z^t, z_{t+1}\right)\right] + \lambda^s\left(z^t\right)$$
$$\lambda^s\left(z^t\right) \cdot \left(a^s\left(z^t\right) - \underline{a}\right) = 0$$
$$a^s\left(z^t\right) - \underline{a} \geq 0$$
$$\lambda^s\left(z^t\right) \geq 0$$

- The generation of terminal age N simply consumes everything it has.

Functional Rational Expectations Equilibrium

See, e.g., Spear (1988)

FREE: A function mapping states to policies that are consistent with the equilibrium conditions.

$$\boxed{\mathbf{f} : \{0,1,2,3\} \times \mathbb{R}^{60} \to \mathbb{R}^{59 \cdot 2}} : \quad \mathbf{f}\left(\begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix}\right) = \mathbf{f}\left(\begin{bmatrix} z_t \\ 0_t^1 \\ k_t^2 \\ \dots \\ k_t^{59} \\ k_t^{60} \end{bmatrix}\right) = \begin{bmatrix} a_t^1 \\ \dots \\ a_t^{59} \\ \lambda_t^1 \\ \dots \\ \lambda_t^{59} \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ \dots \\ a \end{matrix}}\right\} \text{capital investment funct.} \\ \left.\vphantom{\begin{matrix} a \\ \dots \\ a \end{matrix}}\right\} \text{kkt-multiplier borrowing contr.} \end{matrix}$$

such that: $\forall h = 1, \dots, 59$:

$$\left.\begin{aligned} 0 &= \beta \mathrm{E}_t \left[\frac{R_{t+1} u'(c_{t+1}^{h+1}) + \lambda_t^h}{u'(c_t^h)} \right] - 1 \\ 0 &= \lambda_t^h a_t^h \\ 0 &\leq \lambda_t^h \\ 0 &\leq a_t^h \end{aligned}\right\} =: \mathbf{G}\left(\begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix}, \mathbf{f}\left(\begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix}\right)\right)_h$$

$$c_t^h = k_t^h R_t + l_t^h w_t - a_t^h$$

$$R_t = \xi_t \alpha K_t^{\alpha - 1} L_t^{1-\alpha} + (1 - \delta_t)$$

$$w_t = \xi_t (1 - \alpha) K_t^{\alpha} L_t^{-\alpha}$$

$$K_t = \sum_{h=1}^{60} k_t^h$$

$$L_t = \sum_{h=1}^{60} l_t^h$$

2. II. From Neural Networks to “Deep Equilibrium Nets”

- A recursive equilibrium: $\{f_i\}_{i=1}^{N\text{ out}}$, where

$$f_i : \mathbb{R}^{N_{in}} \rightarrow \mathbb{R} : \underbrace{x}_{\text{state}} \rightarrow \qquad f_i(x)$$

- A Deep Equilibrum Net: $\mathcal{N}_\rho$, where

$$\mathcal{N}_\rho : \mathbb{R}^{N_{\text{in}}} \rightarrow \mathbb{R}^{N_{\text{out}}} : \underbrace{\mathbf{x}}_{\text{state}} \rightarrow \boxed{\underbrace{\mathcal{N}_\rho(\mathbf{x})}_{\text{approximate endogenous variables}} \approx \begin{bmatrix} f_1(\mathbf{x}) \\ \cdots \\ f_{N_{\text{out}}}(\mathbf{x}) \end{bmatrix}}$$

What is a deep neural network?

see, e.g., Cybenko (1989), Hornik (1991)

- Neural networks are flexible function approximators.
- A neural net is characterized by its parameters **ρ**.
- Given a parameter vector **ρ** and input vector **x**, denote the neural net as $\mathcal{N}_\rho$, and some desired function with **f**.

$$\boxed{\begin{array}{c}\mathcal{N}_\rho : \mathbb{R}^{N_{\text{in}}} \to \mathbb{R}^{N_{\text{out}}} : \mathbf{x} \to \mathcal{N}_\rho(\mathbf{x}) \\ \mathbf{f}(\mathbf{x}) : \mathbb{R}^{N_{\text{in}}} \to \mathbb{R}^{N_{\text{out}}} : \mathbf{x} \to \mathbf{f}(\mathbf{x})\end{array}}$$

- We desire parameters **ρ**, such that

$$\boxed{\|\mathcal{N}_\rho - \mathbf{f}\|_{\text{some norm}} = 0}$$

What is a deep neural network?

Consider:

$$\begin{aligned}&\textbf{input} := \mathbf{x} \to W_\rho^1 \mathbf{x} + \mathbf{b}_\rho^1 =: \textbf{hidden 1}\\ &\to \textbf{hidden 1} \to W_\rho^2(\textbf{hidden 1}) + \mathbf{b}_\rho^2 =: \textbf{hidden 2}\\ &\to \textbf{hidden 2} \to W_\rho^3(\textbf{hidden 2}) + \mathbf{b}_\rho^3 =: \textbf{output}\end{aligned}$$

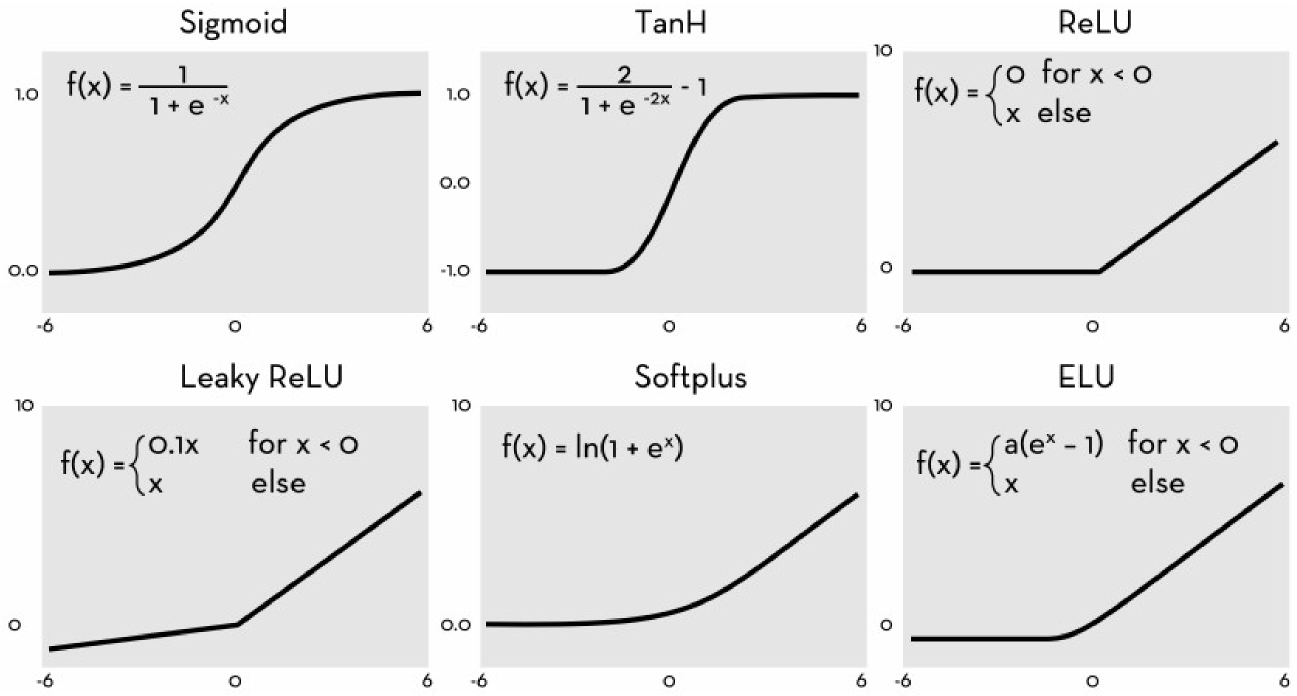
The parameters ρ are the entries of the matrices $\left(W_\rho^1, W_\rho^2, W_\rho^3\right)$ and vectors $\left(\mathbf{b}_\rho^1, \mathbf{b}_\rho^2, \mathbf{b}_\rho^3\right)$

What is a deep neural network?

So far we have a concatenation of affine maps and therefore an affine map.

Next ingredient: **activation functions** ϕ^1, ϕ^2, ϕ^3

Popular activation functions are:



What is a deep neural network?

Now we obtain:

$$\begin{aligned}&\textbf{input} := \mathbf{x} \to \phi^1(W^1_\rho \mathbf{x} + \mathbf{b}^1_\rho) =: \textbf{hidden 1}\\ &\to \textbf{hidden 1} \to \phi^2(W^2_\rho(\textbf{hidden 1}) + \mathbf{b}^2_\rho) =: \textbf{hidden 2}\\ &\to \textbf{hidden 2} \to \phi^3(W^3_\rho(\textbf{hidden 2}) + \mathbf{b}^3_\rho) =: \textbf{output}\end{aligned}$$

The **neural net** is then given by the choice of **activation functions** and the **parameters ρ.**

How to find good parameters ρ?

The standard way:

Step 1: get "labelled data" $\mathcal{D} := \left\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, \left(\mathbf{x}_{|\mathcal{D}|}, \mathbf{y}_{|\mathcal{D}|}\right)\right\}$ where $\mathbf{y}_i = \mathbf{f}(\mathbf{x}_i)$ is the correct output → **Supervised learning**

Step 2: Define a loss function, for example:

$$l_\rho := \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}} \left(\mathbf{y}_i - \mathcal{N}_\rho(\mathbf{x}_i)\right)^2$$

Step 3: Adjust the parameters to minimize the loss via (**stochastic**) **gradient descent**:

$$\rho_i^{\text{new}} = \rho_i^{\text{old}} - \alpha^{\text{step}} \frac{\partial l_{\rho^{\text{old}}}}{\partial \rho_i^{\text{old}}}$$

the step-width α^{step} is called the "**learning rate**" and the process of adjusting the parameters is called "learning".

How to find good parameters ρ?

- The deeper and larger the neural net becomes, the more flexible it is as a function approximator, …

  ... but the more data it will need
  → rule of thumb: **#observations/parameters** ~ 10x (Marsland (2014).

- In economic applications, the standard way of solving for equilibria is to use time iteration → **“small” data**.
  (e.g., Sparse Grids (Brumm & Scheidegger (2017), Krüger & Kübler (2004), Judd et al. (2014))

The issue with time iteration

- **Time Iteration – collocation** (see, e.g., Judd (1998), and references therein)

1. Select a grid G, and a policy function $f^{\text{ start}}$. Set $f^{\text{ next}} \equiv f^{\text{ start}}$.
2. Make one time iteration step:
   1. For all $g \in G$, find $f(g)$ that solves the Period-to-Period Equilibrium Problem given $f^{\text{ next}}$.
   2. Use solutions at all grid points G to interpolate f(how?).
3. Check error criterion: If $\|f - f^{\text{ next}}\|_{\infty} < \epsilon$, report solution: $\tilde{f} = f$
   Else set $f^{\text{ next}} \equiv f$ and go to step 2.

→ Solving non-linear sets of equations in every iteration can be a daunting task.

→ What if non-linear solver does not converge? Construction of surrogate may crash.

→ Can the dynamic economic problem at hand be mapped onto a grid?

2. III. An “economic” loss function

- **Novelty**: we propose an “economic” loss function:

$$l_\rho := \frac{1}{N_{\text{path length}}} \sum_{\mathrm{x_i \text{ on sim. path}}} \left(\mathbf{G}\left(\mathbf{x}_i, \mathcal{N}_\rho\left(\mathbf{x}_i\right)\right)\right)^2$$

where we use $\mathcal{N}_\rho$ to simulate a path.

- **G** is chosen such that the **true equilibrium policy f(x)** is defined by

$$\boxed{\mathbf{G}(\mathbf{x}, \mathbf{f}(\mathbf{x})) = 0 \ \forall \mathbf{x}.}$$

- **G(.,.): implied error in the optimality conditions** (unit-free Euler errors)

- Therefore, there is **no need for labels** to evaluate our loss function.
 →**Unsupervised Machine Learning**.

Recall OLG: an explicit cost function

$$\ell_{\mathcal{D}_{\text{train}}}(\rho) := \frac{1}{|\mathcal{D}_{\text{train}}|}\frac{1}{N-1}\sum_{\mathbf{x}_j \in \mathcal{D}_{\text{train}}}\sum_{i=1}^{N-1}\left(\left(e^i_{\text{REE}}(\mathbf{x}_j)\right)^2 + \left(e^i_{\mathbf{KKT}}(\mathbf{x}_j)\right)^2\right)$$

$$e^i_{\text{REE}}(\mathrm{x}_j) := \frac{u'^{-1}\left(\beta \mathrm{E}_{z_j}\left[r\left(\hat{\mathbf{x}}_{j,+}\right)u'\left(\hat{c}^{i+1}\left(\hat{\mathbf{x}}_{j,+}\right)\right)\right] + \hat{\lambda}^i(\mathbf{x}_j)\right)}{\hat{c}^i(\mathbf{x}_j)} - 1$$

$$e^i_{\mathbf{KKT}}(\mathbf{x}_j) := \hat{\lambda}^i(\mathbf{x}_j)\left(\hat{a}^i(\mathbf{x}_j) - \underline{a}\right)$$

$$\hat{\mathbf{x}}_{j,+} = \begin{bmatrix} z_+ \\ 0 \\ \hat{a}^{[1:N-1]}(\mathbf{x}_j) \end{bmatrix} \longrightarrow$$

Sampling from the relevant states

Training Deep Equilibrium Nets

Algorithm 1: Algorithm for training deep equilibrium nets.

Data:
T (length of an episode),
N^{epochs} (number of epochs on each episode),
$\tau^{\max}$ (desired threshold for max error),
τ^{mean} (desired threshold for mean error),
$\epsilon^{\text{mean}} = \infty$ (starting value for current mean error),
$\epsilon^{\max} = \infty$ (starting value for current max error),
N^{iter} (maximum number of iterations),
$\boldsymbol{\rho}^0$ (initial parameters of the neural network),
$\mathbf{x}_1^0$ (initial state to start simulations from),
$i = 0$ (set iteration counter),
α^{learn} (learning rate)

Result:
success (boolean if thresholds were reached)
$\boldsymbol{\rho}^{\text{final}}$ (final neural network parameters)

while $((i < N^{iter}) \wedge ((\epsilon^{mean} \geq \tau^{mean}) \vee (\epsilon^{max} \geq \tau^{max})))$ **do**
  $\mathcal{D}^i_{\text{train}} \leftarrow \{\mathbf{x}_1^i, \mathbf{x}_2^i, \ldots, \mathbf{x}_T^i\}$ (generate new training data by simulating an episode of T periods as implied by the parameters $\boldsymbol{\rho}^i$)
  $\mathbf{x}_0^{i+1} \leftarrow \mathbf{x}_T^i$ (set new starting point)
  $\epsilon_{\max} \leftarrow \max\left\{\max_{\mathbf{x}\in\mathcal{D}^i_{\text{train}}} |e_{\mathbf{x}}^{\cdots}(\boldsymbol{\rho})|\right\}$ (calculate max error on new data)
  $\epsilon_{\text{mean}} \leftarrow \max\left\{\frac{1}{T}\sum_{\mathbf{x}\in\mathcal{D}^i_{\text{train}}} |e_{\mathbf{x}}^{\cdots}(\boldsymbol{\rho})|\right\}$ (calculate mean error on new data)
  for $j \in [1, ..., N^{epochs}]$ **do**
    (learn N^{epochs} on data)
    for $k \in [1, ..., length(\boldsymbol{\rho})]$ **do**

$$\rho_k^{i+1} = \rho_k^i - \alpha^{\text{learn}} \frac{\partial \ell_{\mathcal{D}^i_{\text{train}}}(\boldsymbol{\rho}^i)}{\partial \rho_k^i}$$

      (do a gradient descent step to update the network parameters)
    end
  end
  $i \leftarrow i + 1$ (update episode counter)
end
if $i = N^{iter}$ **then return** (success $\leftarrow False, \boldsymbol{\rho}^{\text{final}} \leftarrow \boldsymbol{\rho}^i$) ;
else return (success $\leftarrow True, \boldsymbol{\rho}^{\text{final}} \leftarrow \boldsymbol{\rho}^i$) ;

Deep Equilibrium Net – Architecture

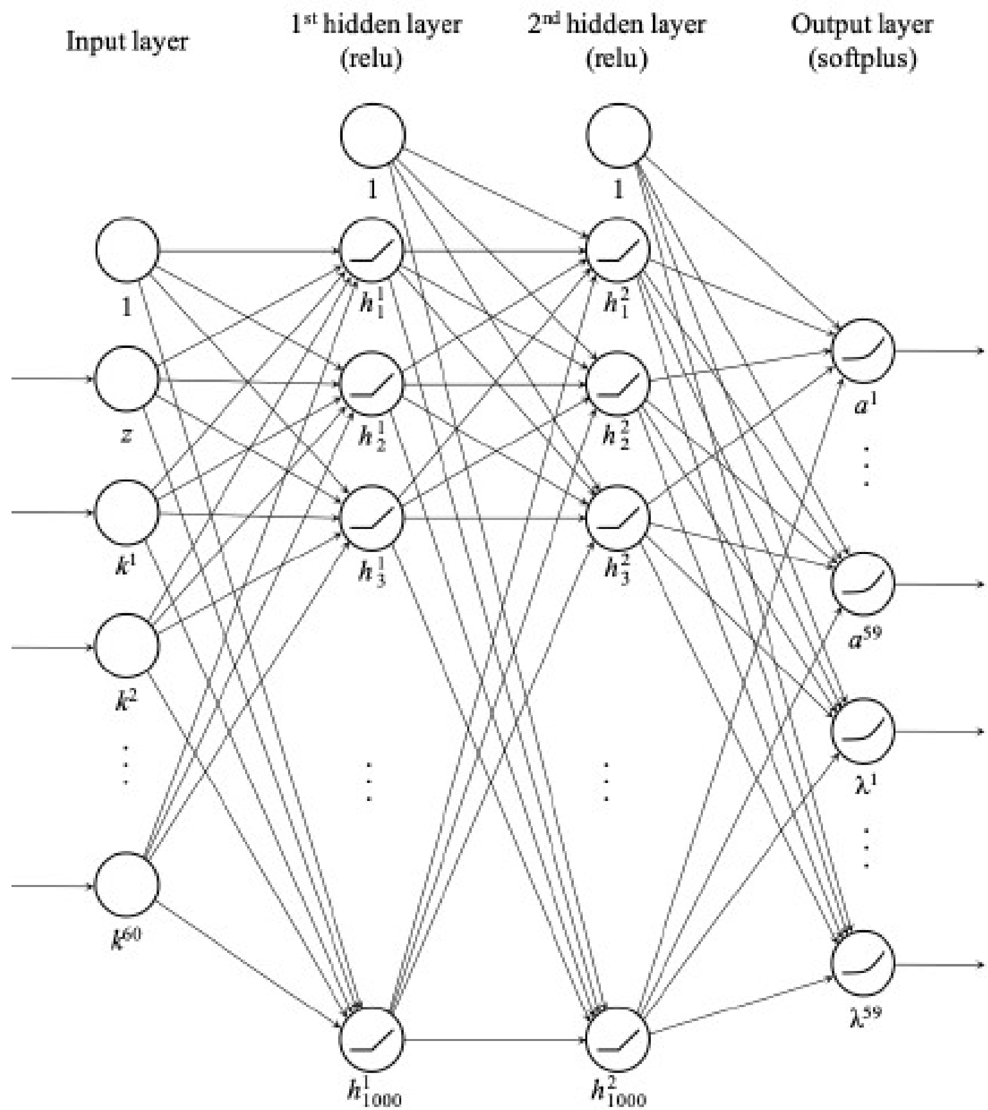
$$\mathcal{N}_{\rho} : \{0,1,2,3\} \times \mathbb{R}^{60} \to \mathbb{R}^{59 \cdot 2} :$$

$$\mathcal{N}_{\rho}\left(\begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix}\right) = \begin{bmatrix} a_t^1 \\ \cdots \\ a_t^{59} \\ \lambda_t^1 \\ \cdots \\ \lambda_t^{59} \end{bmatrix}$$

such that

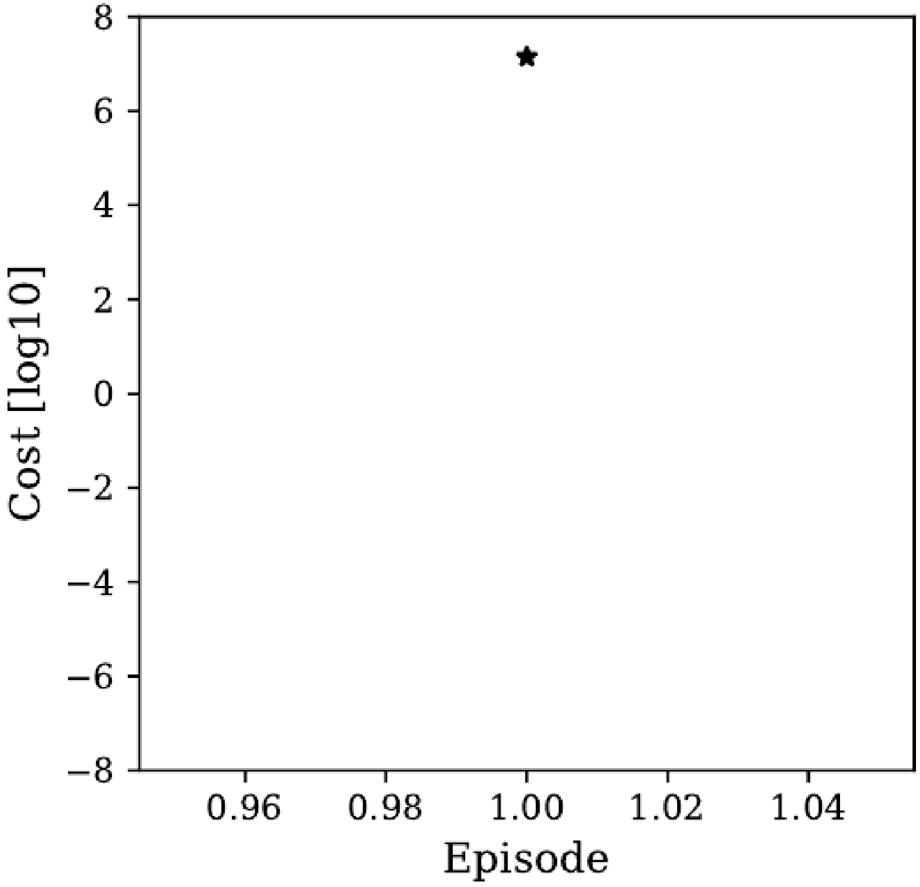
$$\mathbf{G}\left(\begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix}, \mathcal{N}_{\rho}\left(\begin{bmatrix} z_t \\ \mathbf{k}_t \end{bmatrix}\right)\right) \approx \mathbf{0}$$

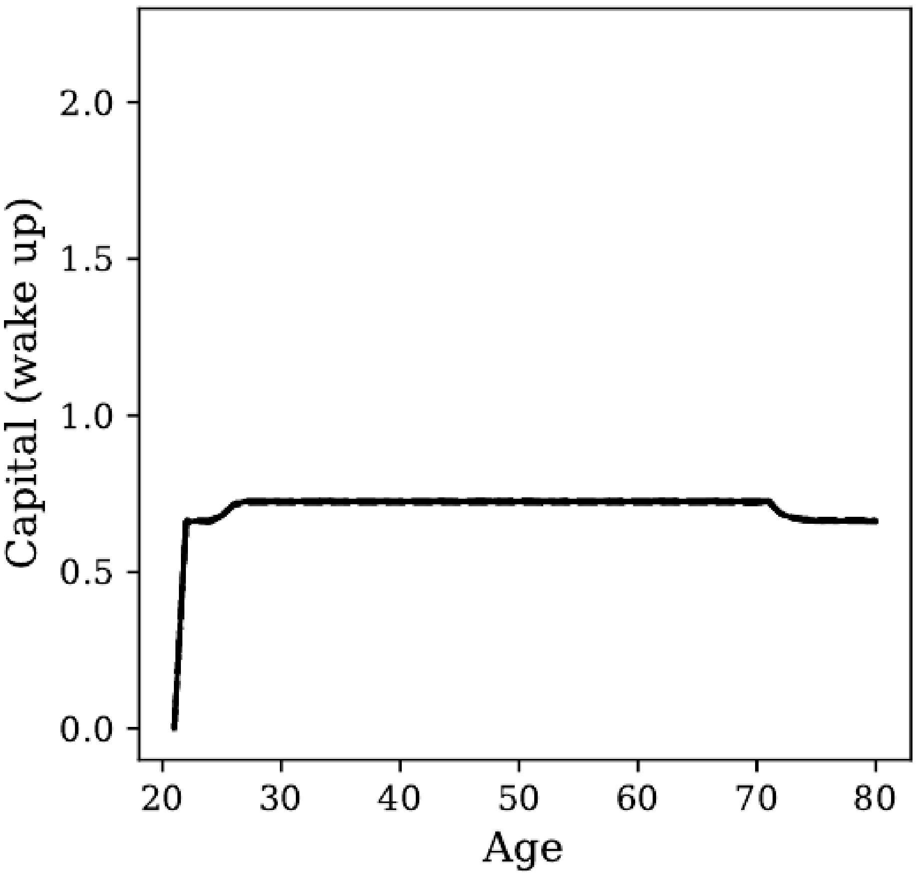
$$\hat{\mathbf{x}}_+ = \begin{bmatrix} z_+ \\ 0 \\ \hat{a}^{[1:N-1]}(\mathbf{x}) \end{bmatrix}$$



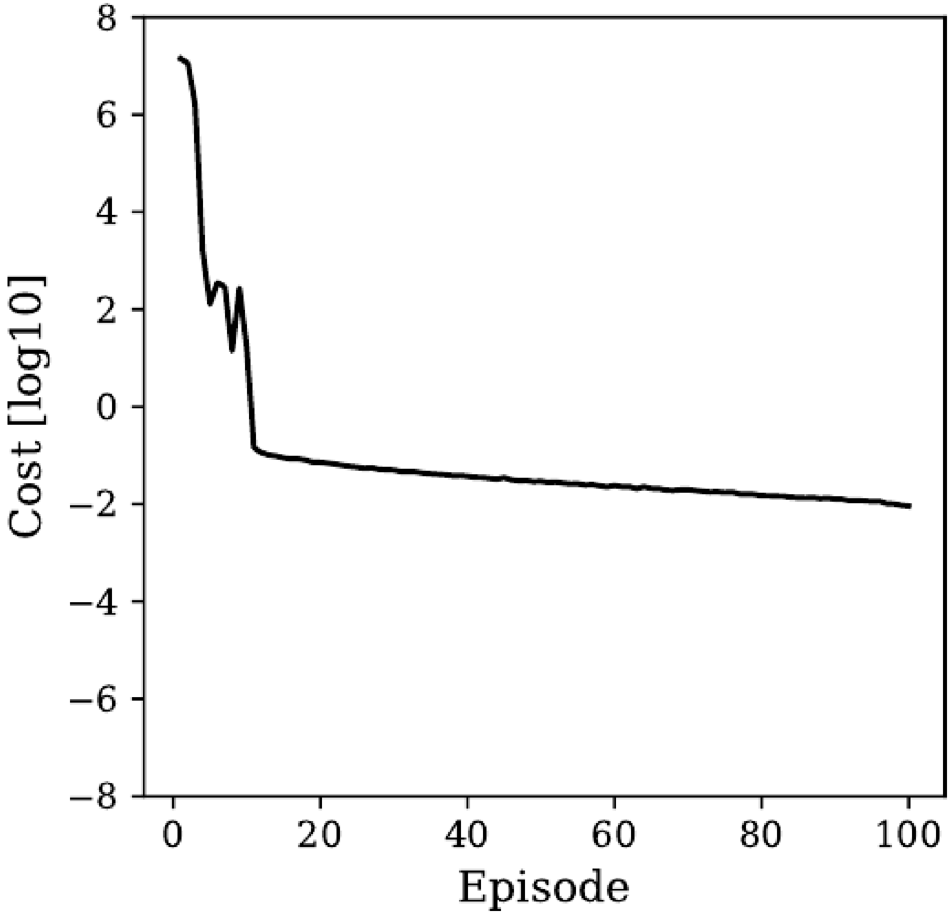
Learning the Equilibrium

Def.: **Episode**: the set of T simulated periods.
Epoch: when the whole dataset is passed through the algorithm.
Per epoch, the neural network parameters are updated T /m times.

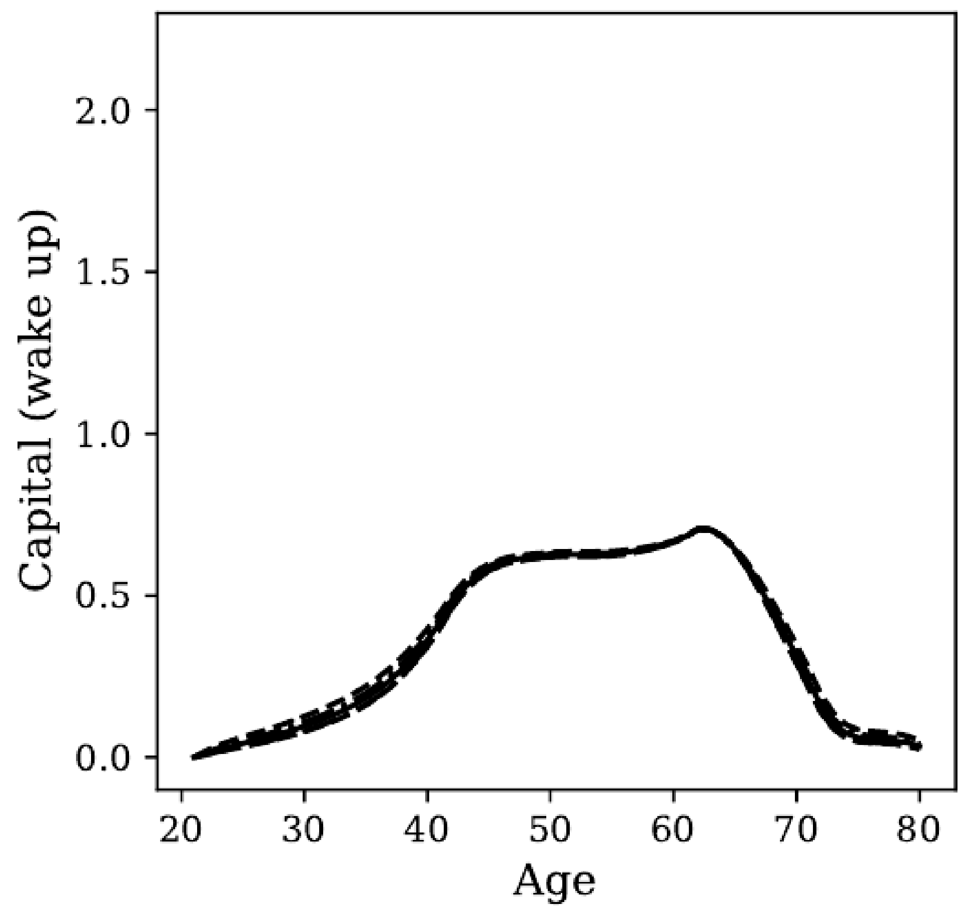




Learning the Equilibrium



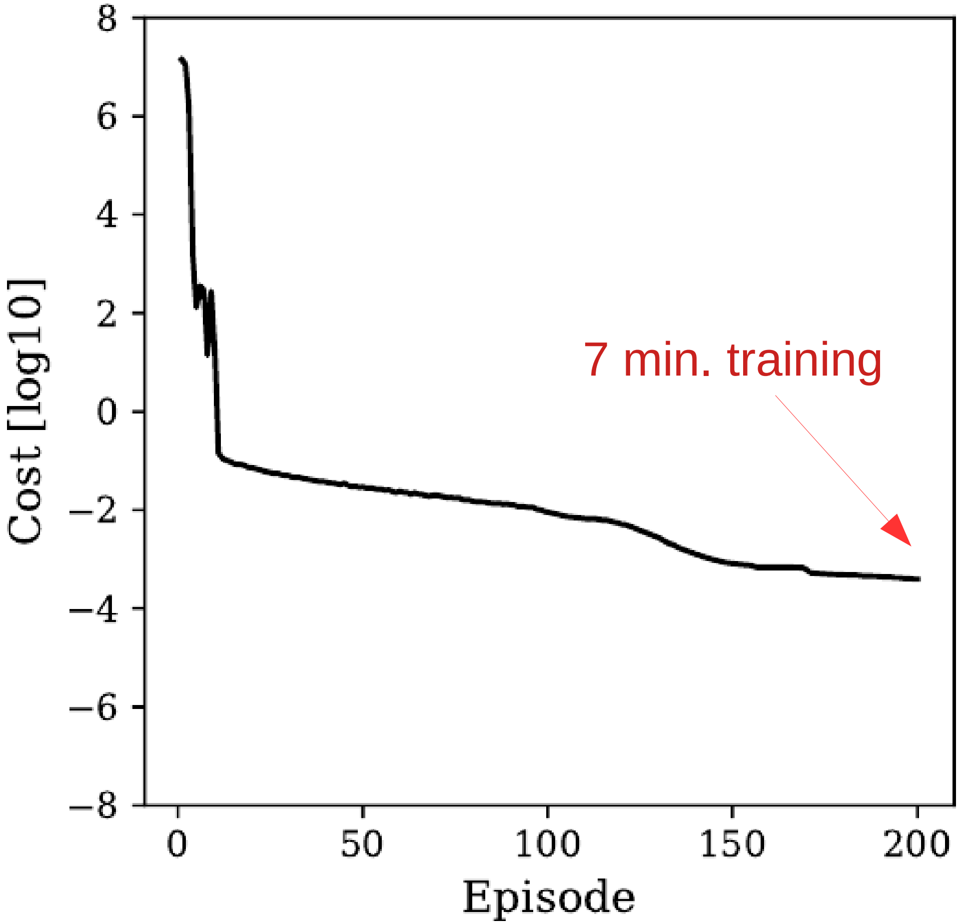
Typical runtime: 2.1 [sec/Episode]



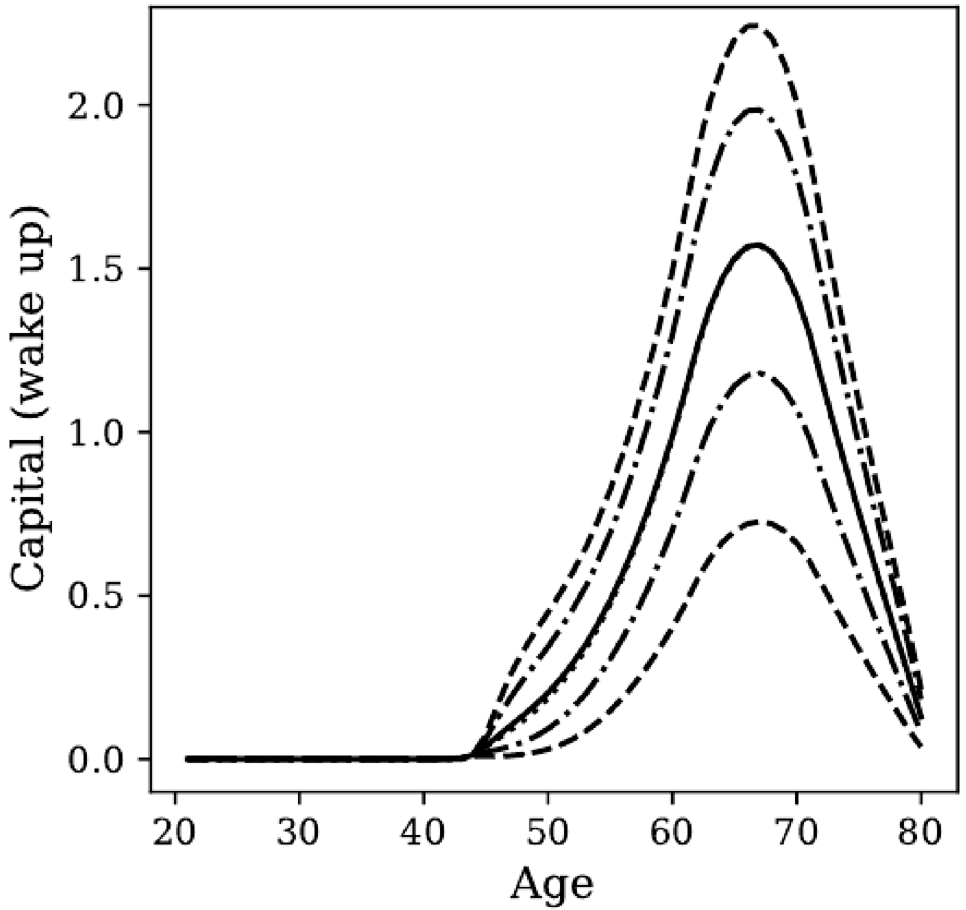
Learning rate: 10^{-5}
Batch size: 1,000; 10,000 periods/Episode

Learning the Equilibrium

Solid line: **mean** over 10,000 simulated periods
Dash-dotted lines show the **10th and 90th percentile**
Dashed lines show the **0.1th and 99.9th percentile**.



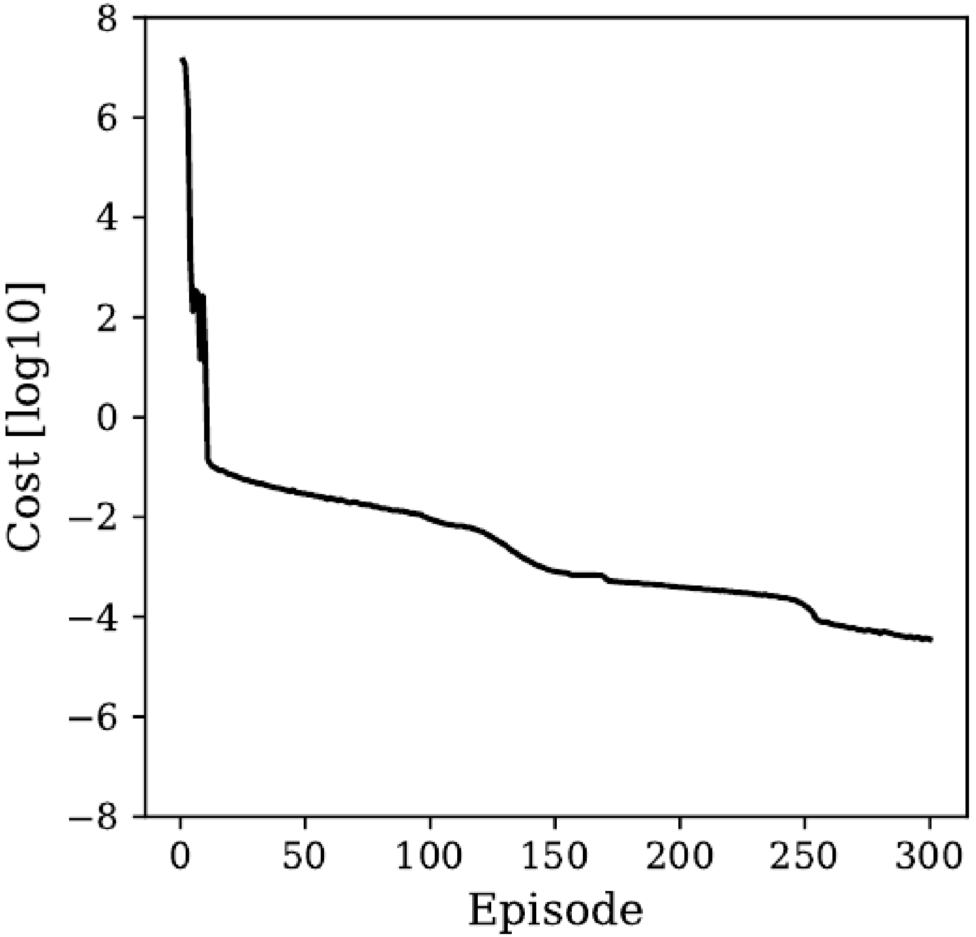
Typical runtime: 2.1 [sec/Episode]



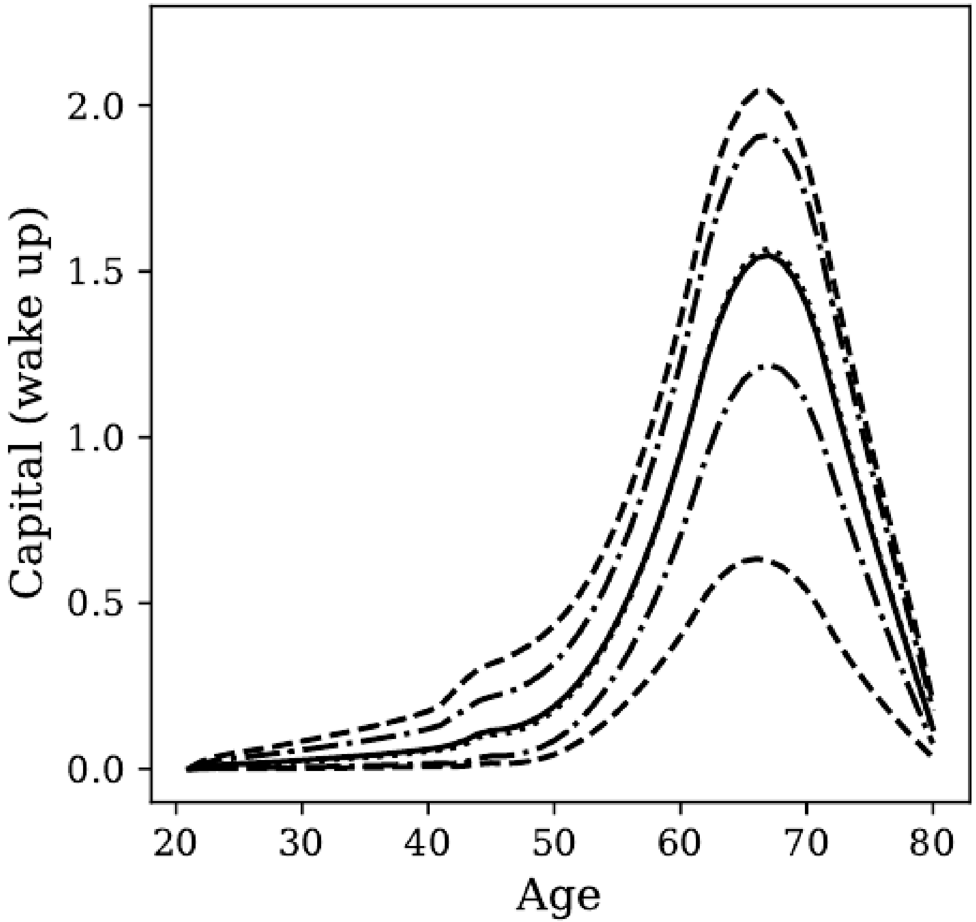
Learning rate: 10^{-5}
Batch size: 1,000; 10,000 periods/Episode

Learning the Equilibrium

Solid line: **mean** over 10,000 simulated periods
Dash-dotted lines show the **10th and 90th percentile**
Dashed lines show the **0.1th and 99.9th percentile**.

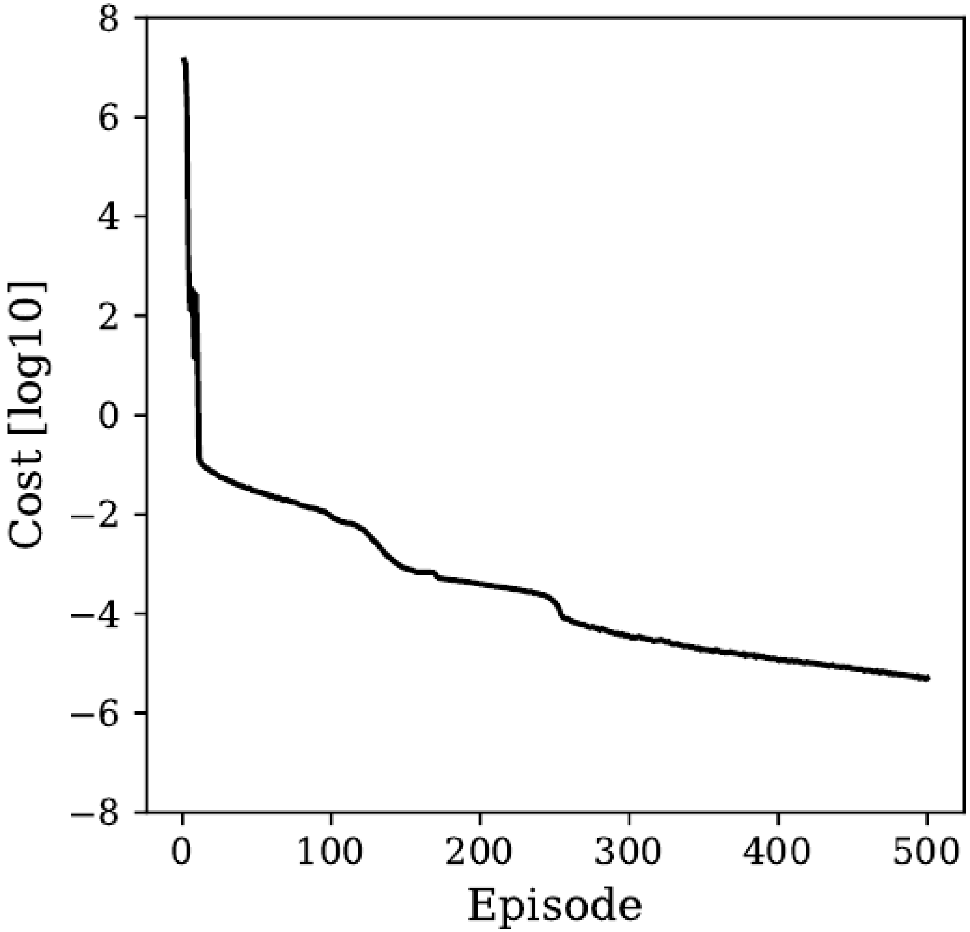


Typical runtime: 2.1 [sec/Episode]



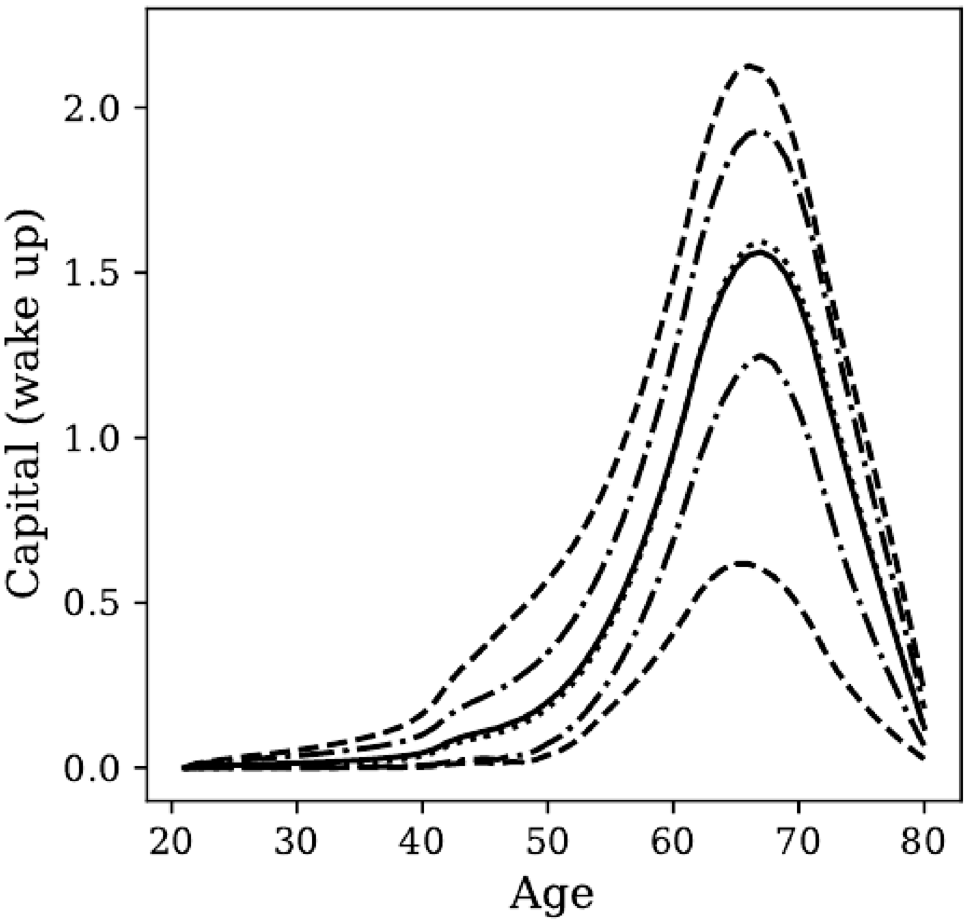
Learning rate: 10^{-5}
Batch size: 1,000; 10,000 periods/Episode

Learning the Equilibrium



Typical runtime: 2.1 [sec/Episode]

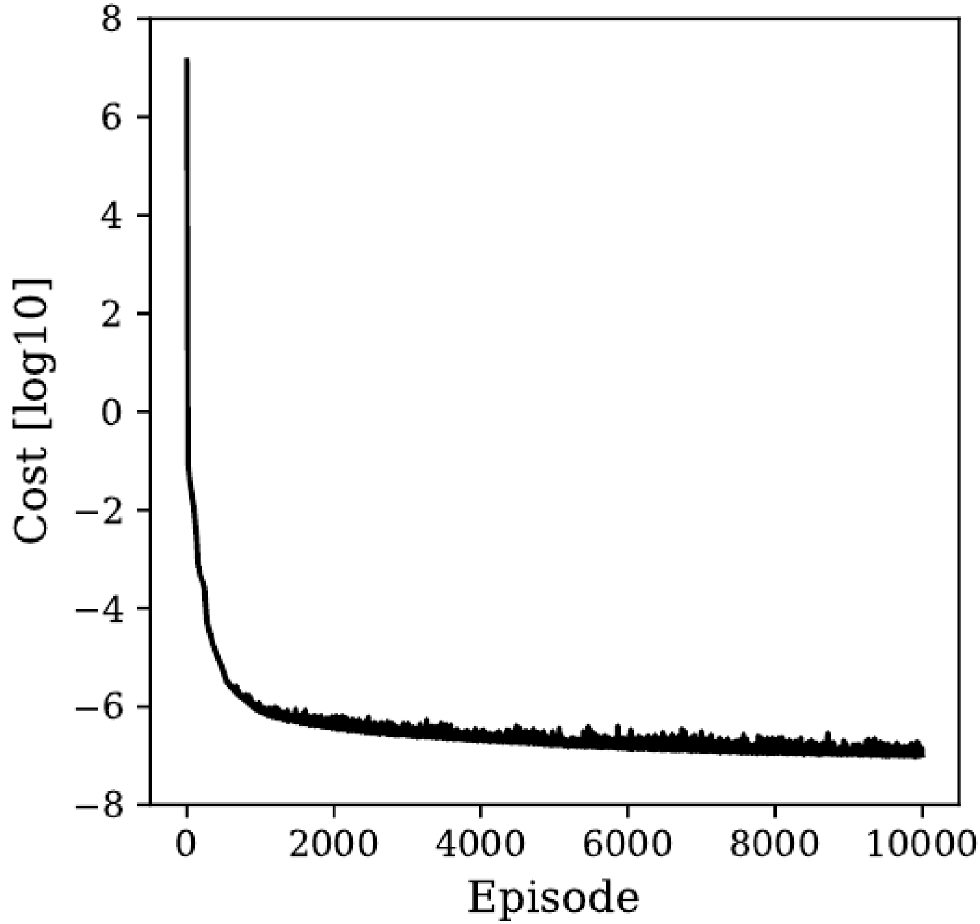
Solid line: **mean** over 10,000 simulated periods
Dash-dotted lines show the **10th and 90th percentile**
Dashed lines show the **0.1th and 99.9th percentile**.



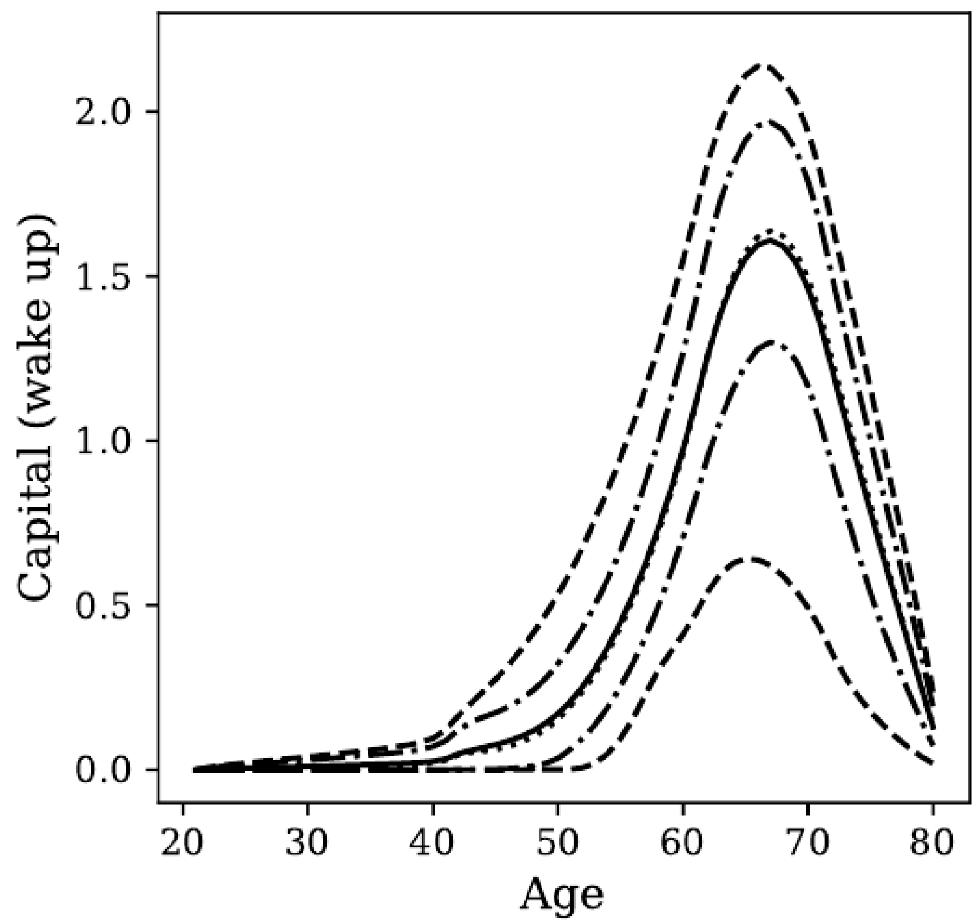
Learning rate: 10^{-5}
Batch size: 1,000; 10,000 periods/Episode

Learning the Equilibrium

Solid line: **mean** over 10,000 simulated periods
Dash-dotted lines show the **10th and 90th percentile**
Dashed lines show the **0.1th and 99.9th percentile**.

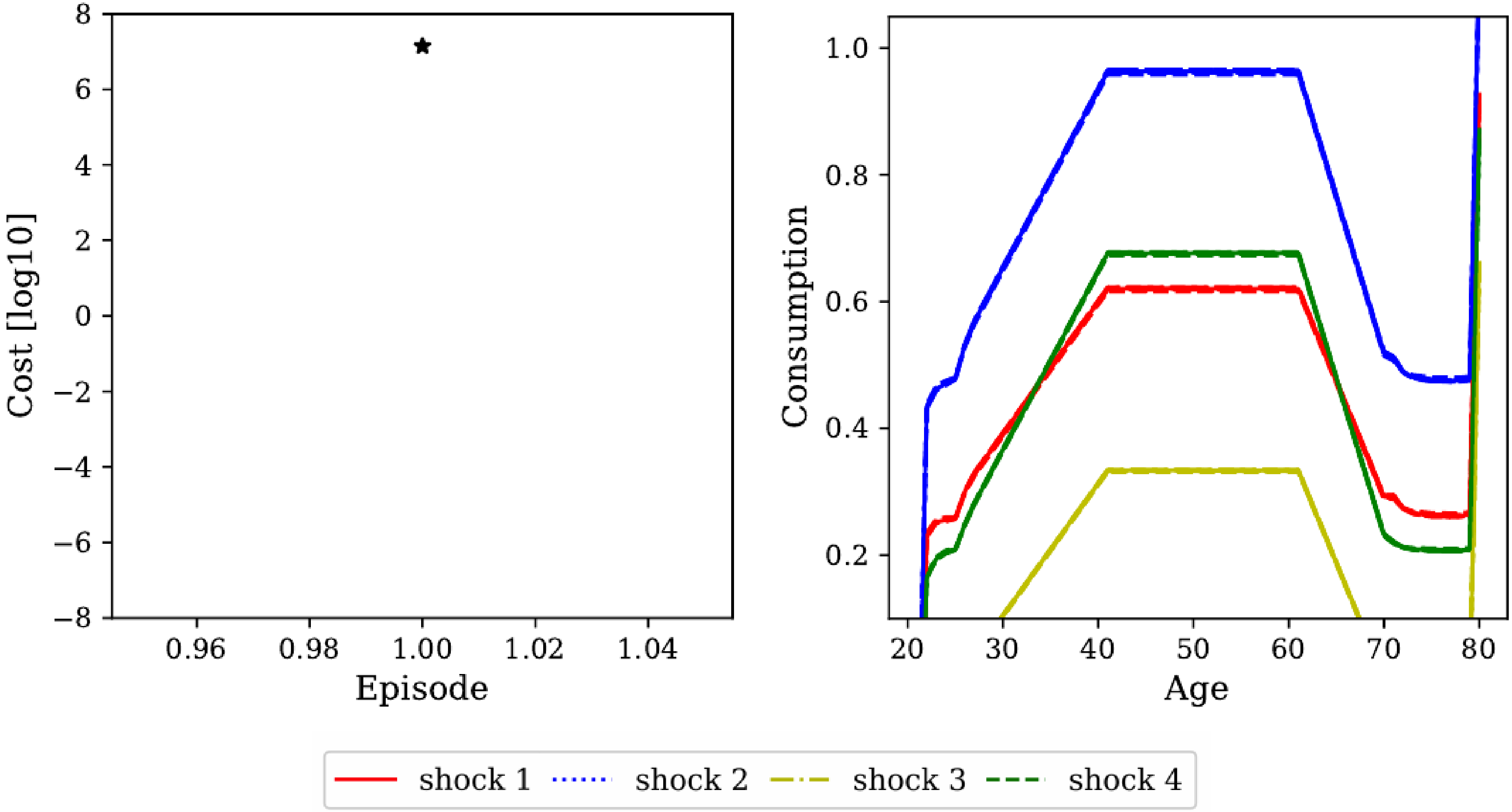


Typical runtime: 2.1 [sec/Episode]



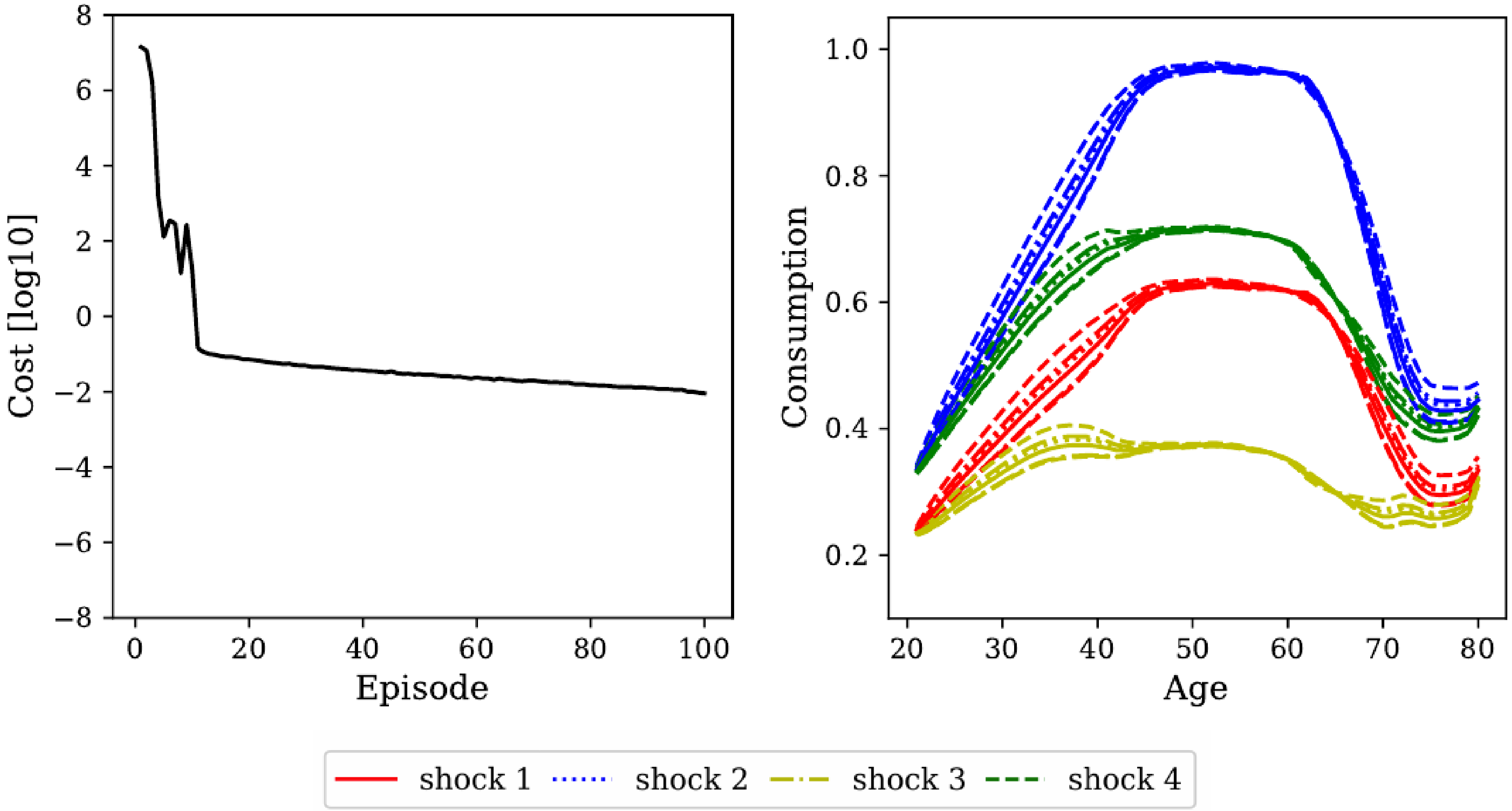
Learning rate: 10^{-5}
Batch size: 1,000; 10,000 periods/Episode

Learning the Equilibrium



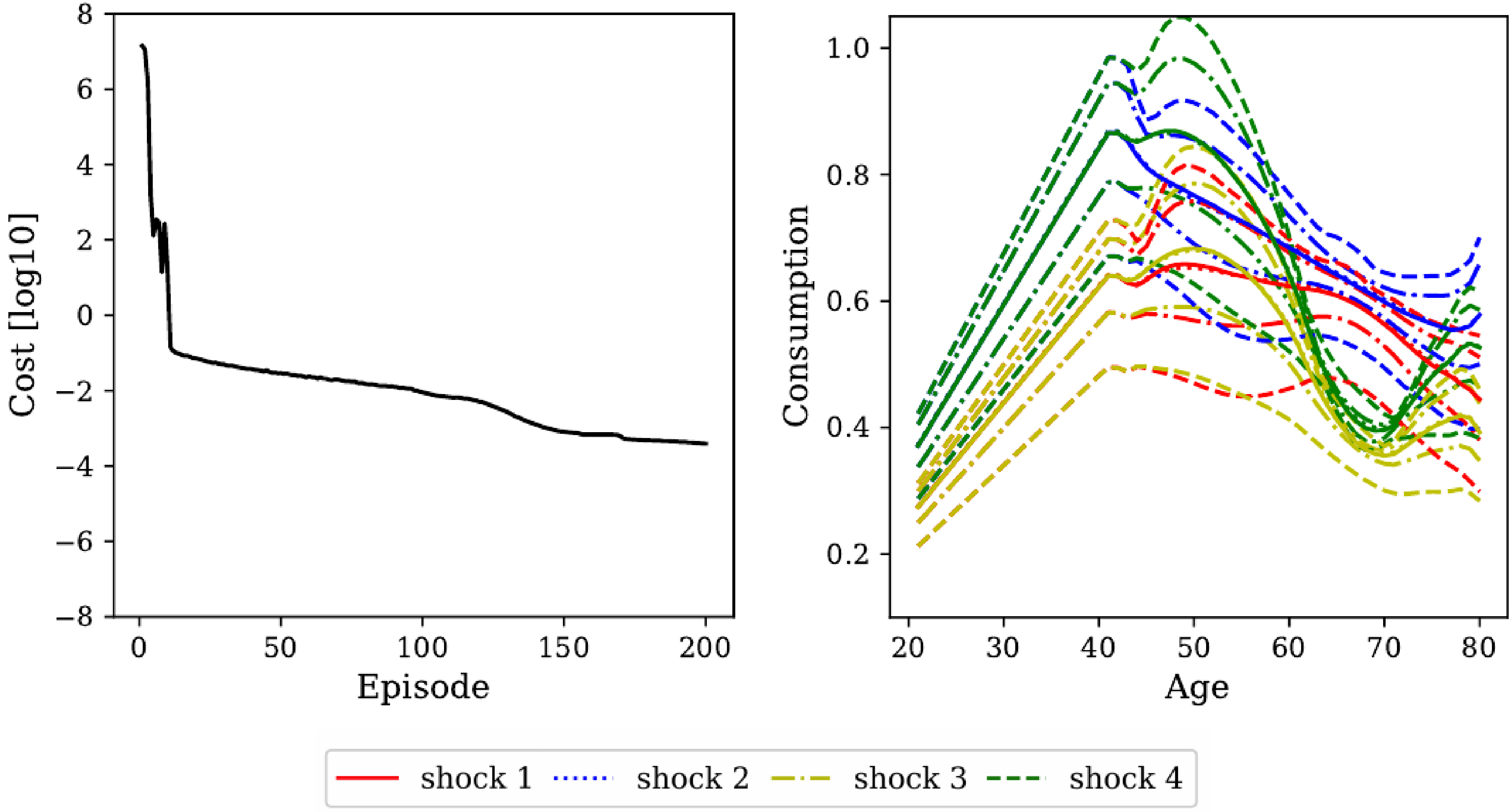
shock 1: $\delta = 0.5,\ \xi = 0.85$, shock 2: $\delta = 0.5,\ \xi = 1.15$, shock 3: $\delta = 0.9,\ \xi = 0.85$, shock 4: $\delta = 0.9,\ \xi = 1.15$

Learning the Equilibrium



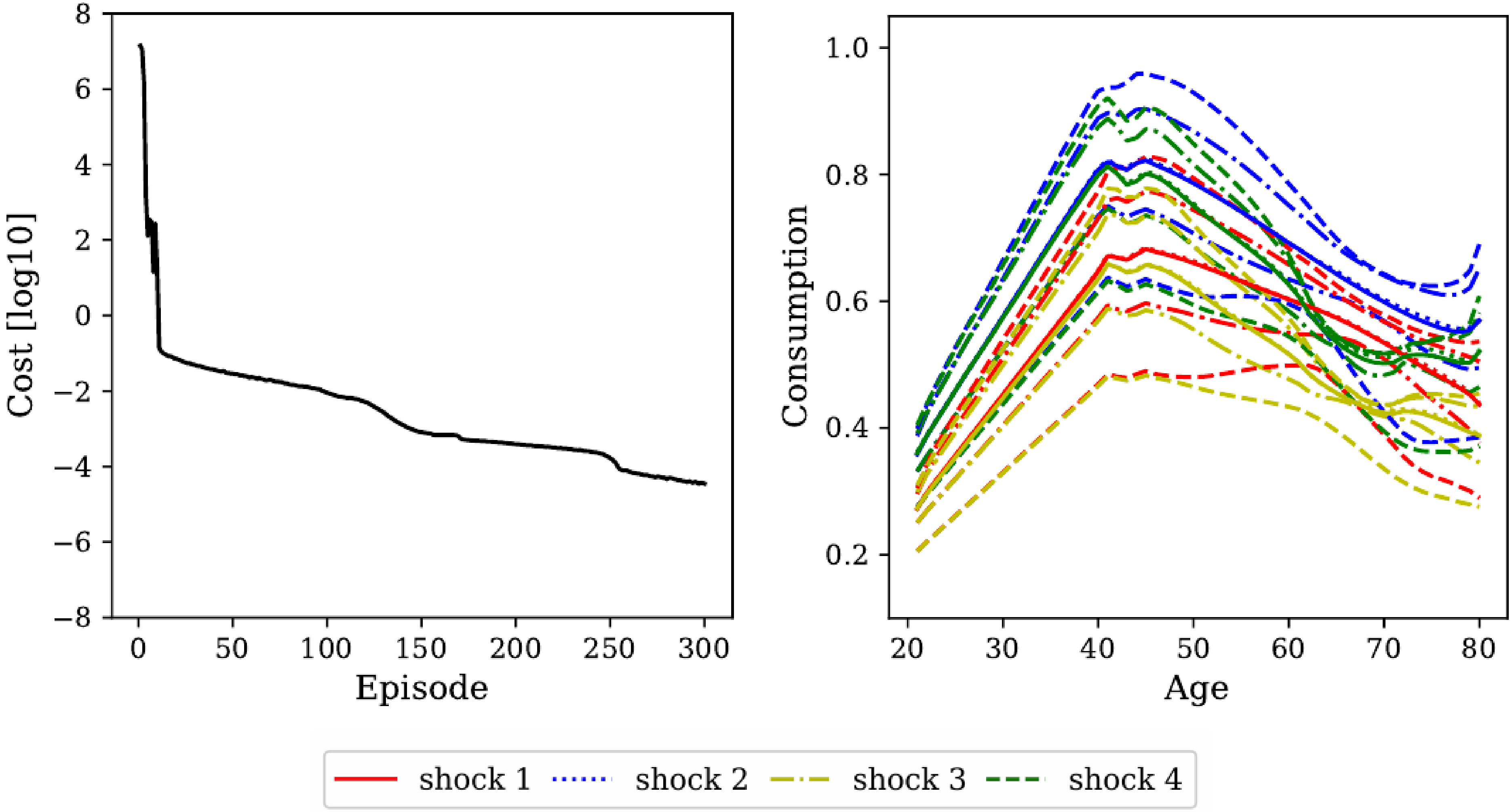
shock 1: $\delta = 0.5,\ \xi = 0.85$, shock 2: $\delta = 0.5,\ \xi = 1.15$, shock 3: $\delta = 0.9,\ \xi = 0.85$, shock 4: $\delta = 0.9,\ \xi = 1.15$

Learning the Equilibrium



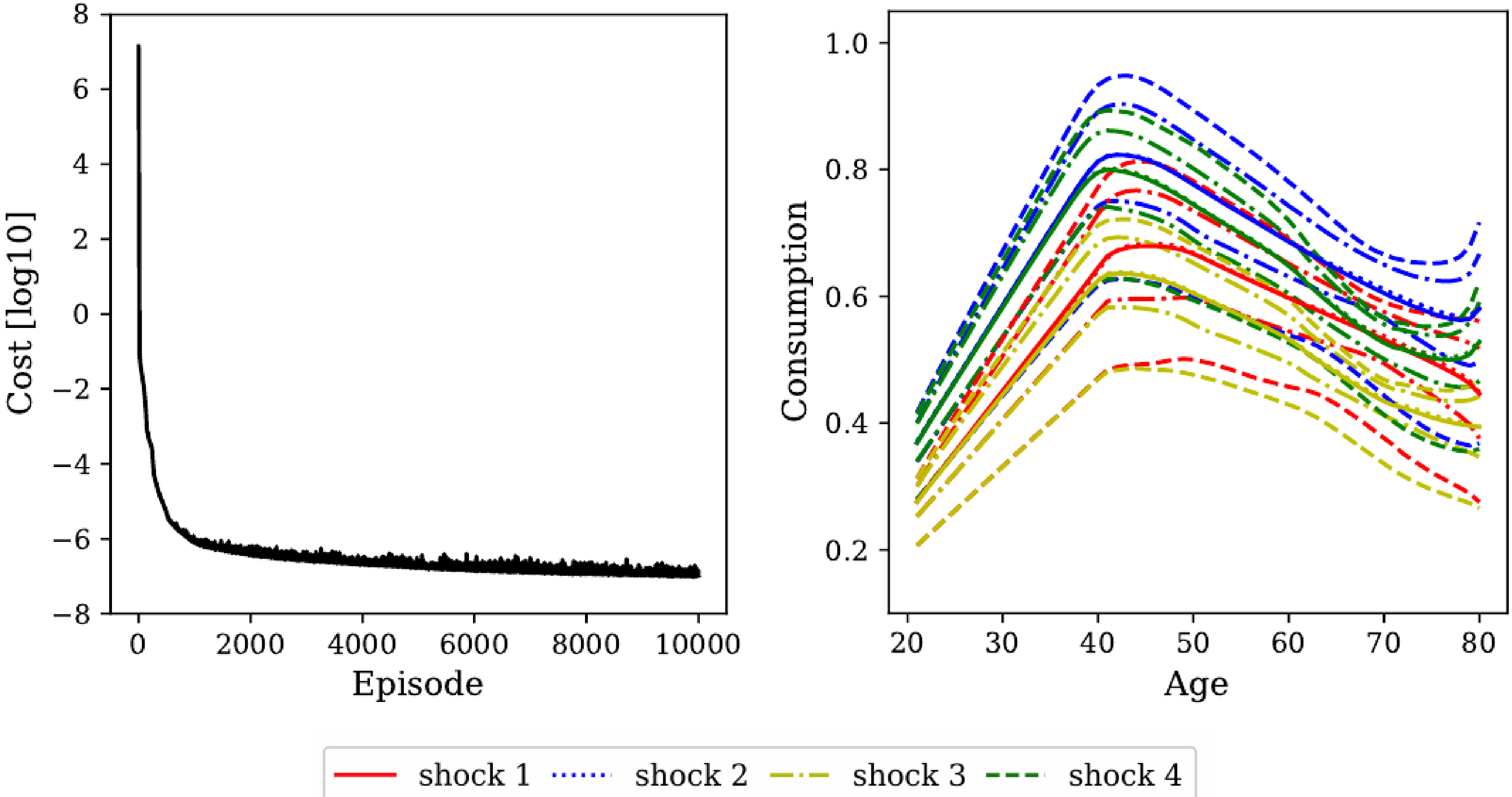
shock 1: $\delta = 0.5,\ \xi = 0.85$, shock 2: $\delta = 0.5,\ \xi = 1.15$, shock 3: $\delta = 0.9,\ \xi = 0.85$, shock 4: $\delta = 0.9,\ \xi = 1.15$

Learning the Equilibrium



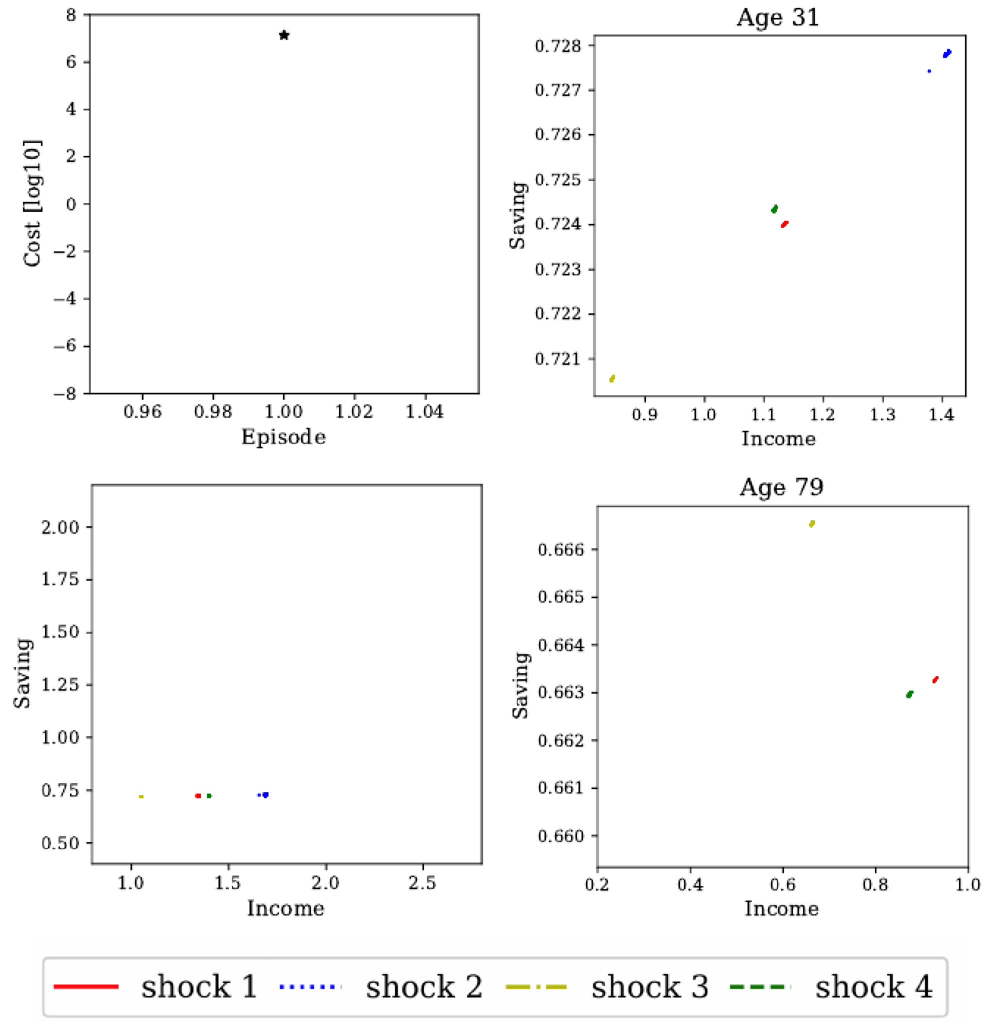
shock 1: $\delta = 0.5,\ \xi = 0.85$, shock 2: $\delta = 0.5,\ \xi = 1.15$, shock 3: $\delta = 0.9,\ \xi = 0.85$, shock 4: $\delta = 0.9,\ \xi = 1.15$

Learning the Equilibrium



shock 1: $\delta = 0.5,\ \xi = 0.85$, shock 2: $\delta = 0.5,\ \xi = 1.15$, shock 3: $\delta = 0.9,\ \xi = 0.85$, shock 4: $\delta = 0.9,\ \xi = 1.15$

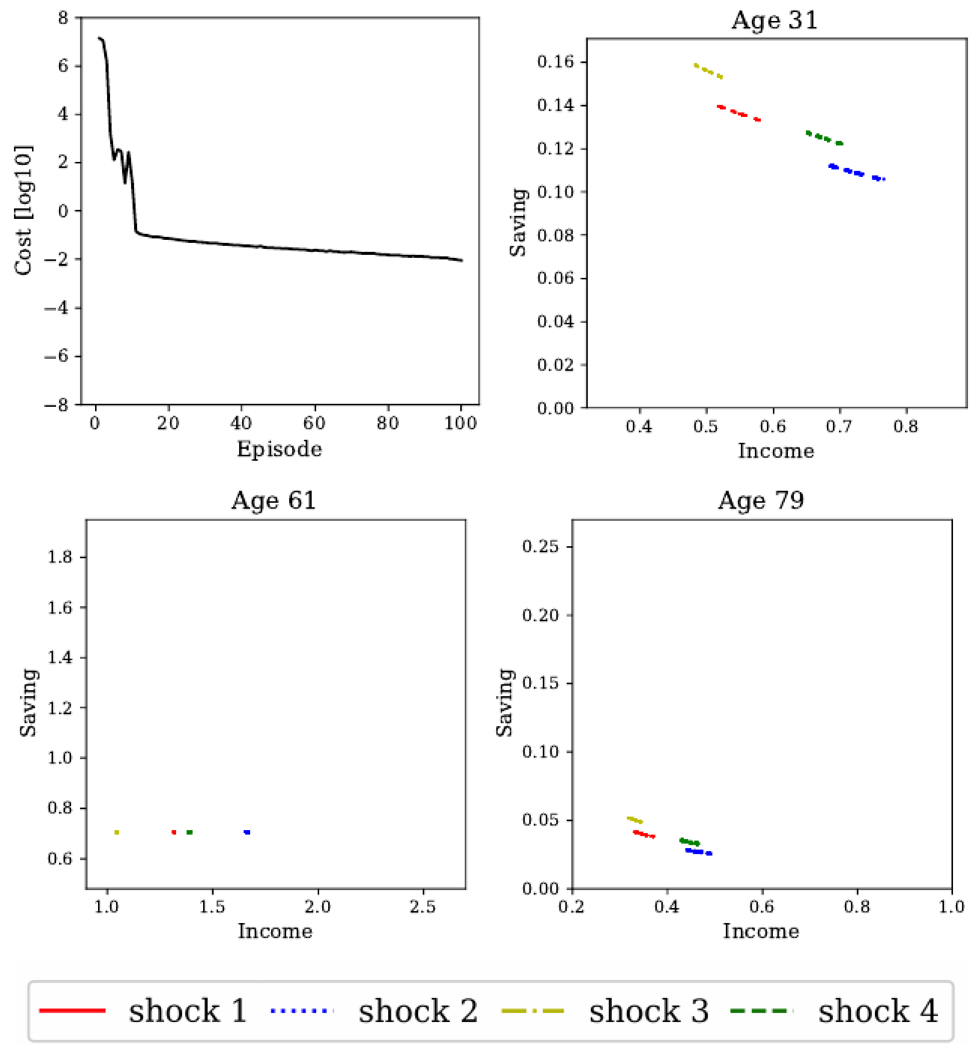
Learning the Equilibrium



Saving: in capital
$w * l + k * r$: Income
$k * r$: financial wealth
$w * l$: labor income

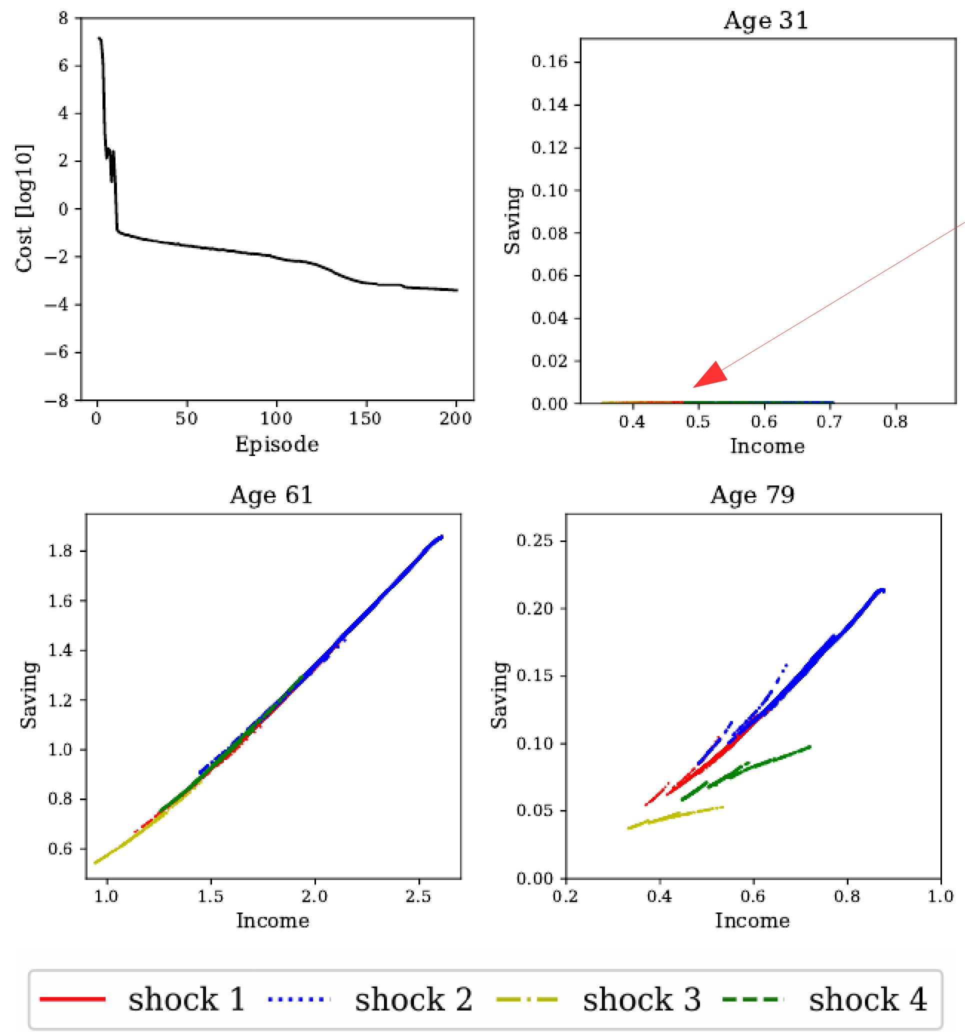
shock 1 : $\delta = 0.5, \xi = 0.85$, shock 2 : $\delta = 0.5, \xi = 1.15$, shock 3 : $\delta = 0.9, \xi = 0.85$ shock 4 : $\delta = 0.9, \xi = 1.15$

Learning the Equilibrium



shock $1 : \delta = 0.5, \xi = 0.85$, shock $2 : \delta = 0.5, \xi = 1.15$, shock $3 : \delta = 0.9, \xi = 0.85$ shock $4 : \delta = 0.9, \xi = 1.15$

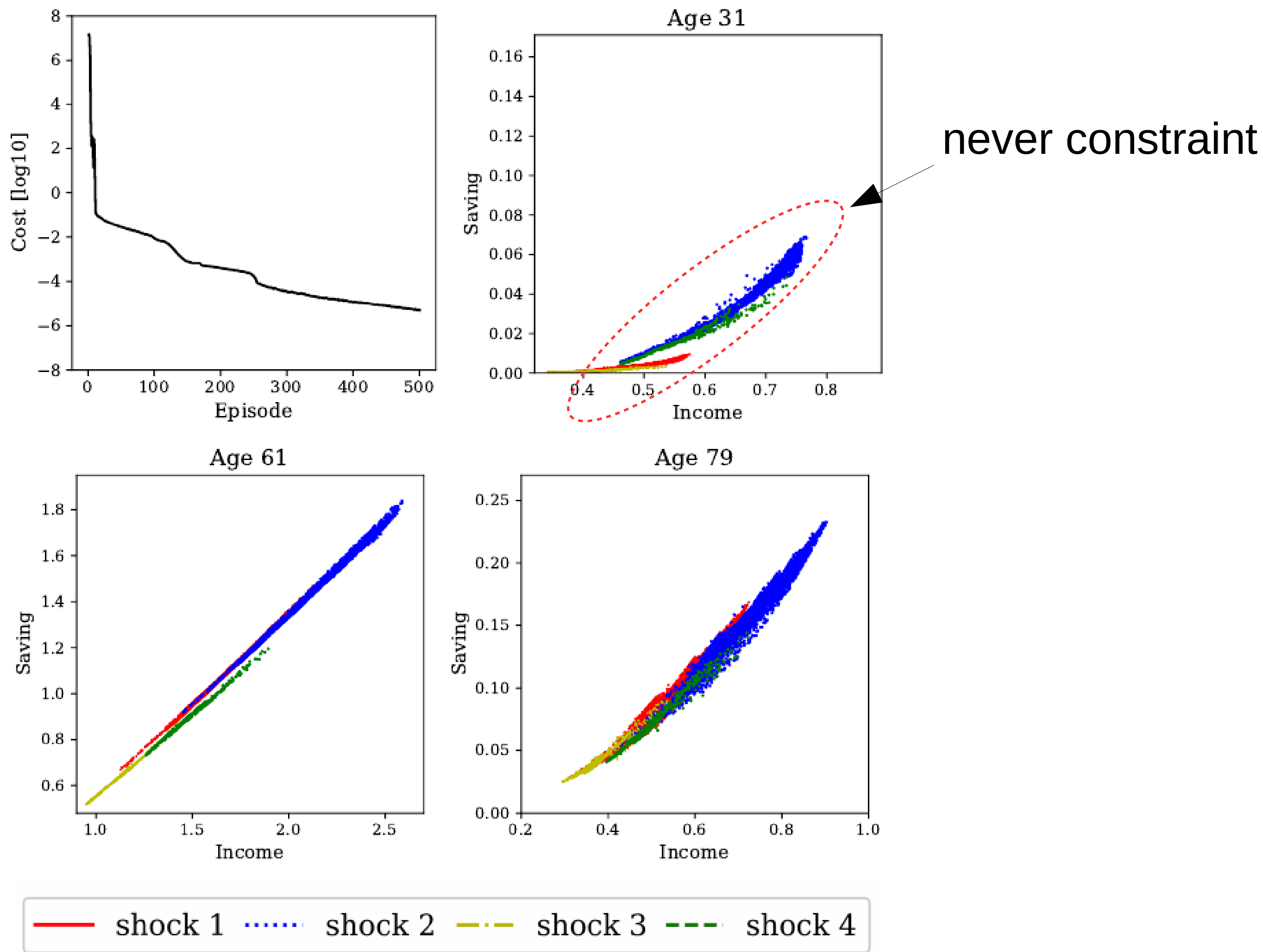
Learning the Equilibrium



always constraint

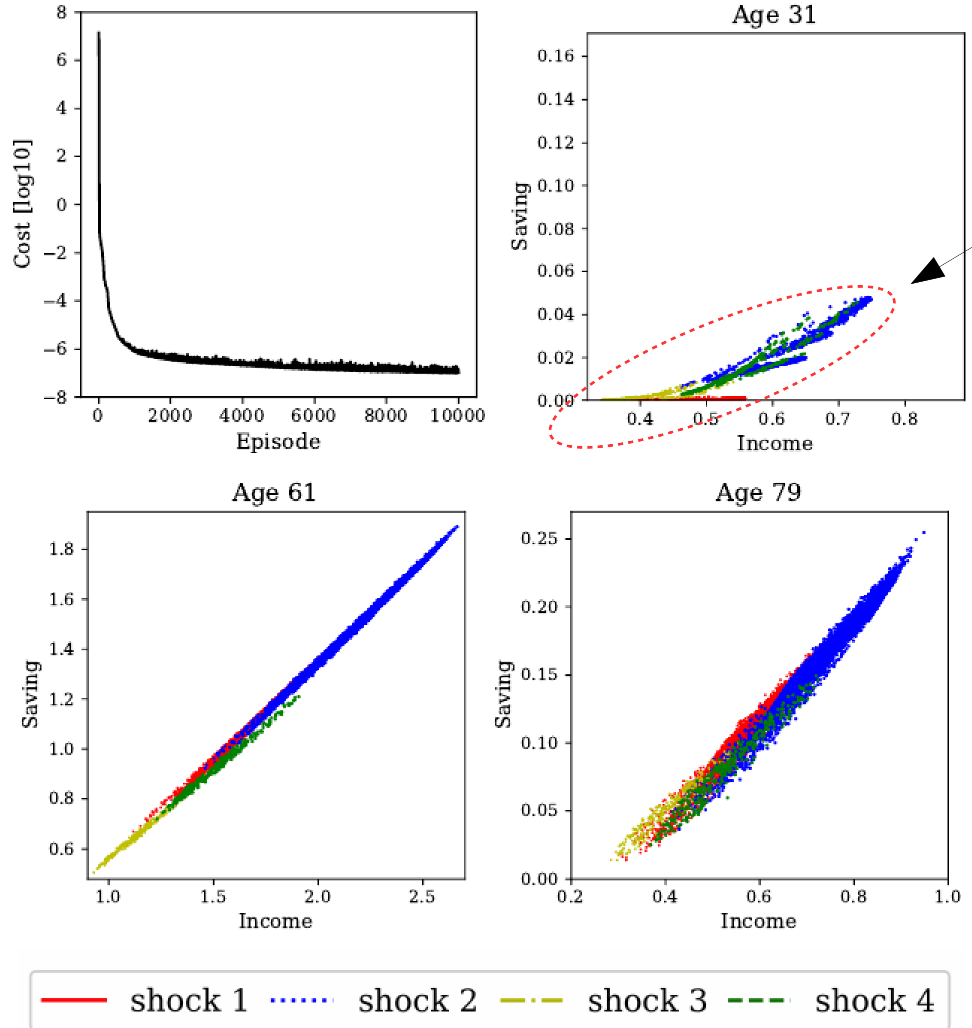
shock $1 : \delta = 0.5, \xi = 0.85$, shock $2 : \delta = 0.5, \xi = 1.15$, shock $3 : \delta = 0.9, \xi = 0.85$ shock $4 : \delta = 0.9, \xi = 1.15$

Learning the Equilibrium



shock 1 : $\delta = 0.5, \xi = 0.85$, shock 2 : $\delta = 0.5, \xi = 1.15$, shock 3 : $\delta = 0.9, \xi = 0.85$ shock 4 : $\delta = 0.9, \xi = 1.15$

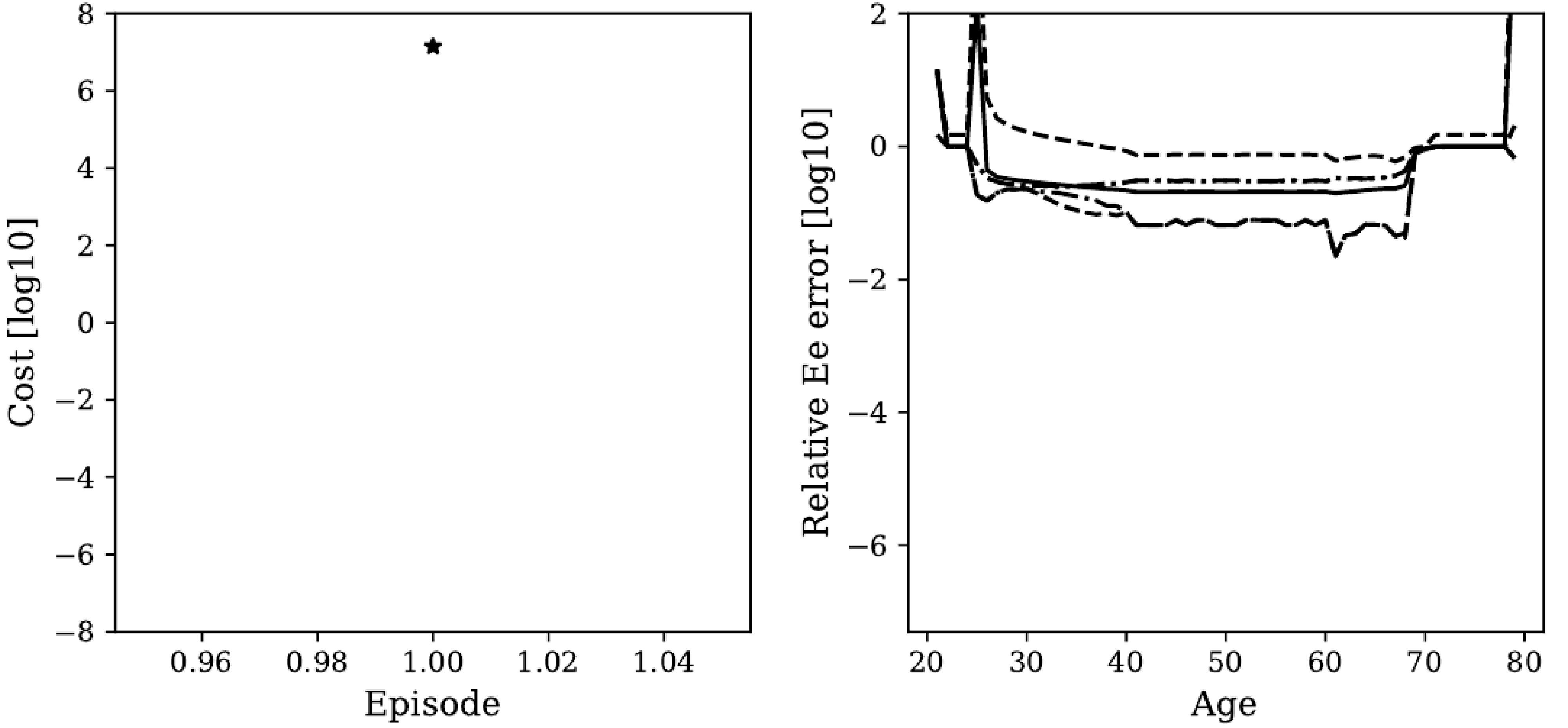
Learning the Equilibrium



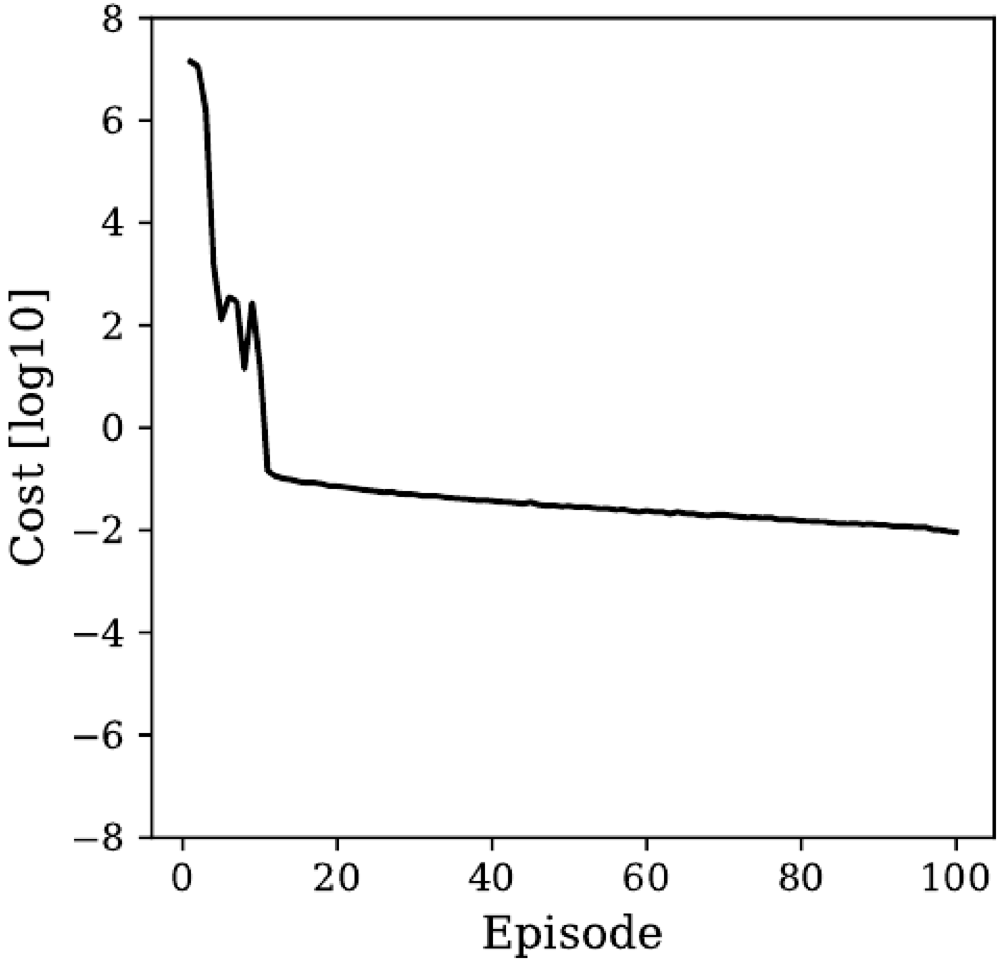
sometimes constraint

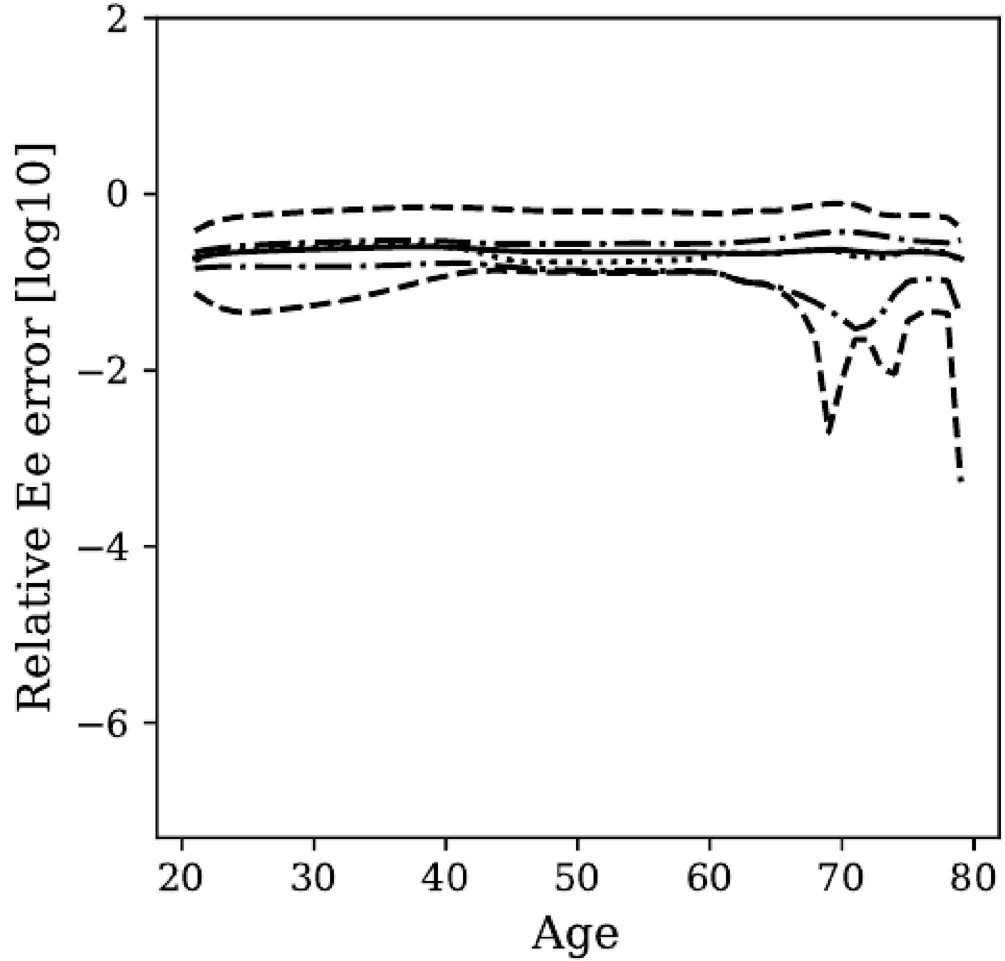
shock 1 : $\delta = 0.5, \xi = 0.85$, shock 2 : $\delta = 0.5, \xi = 1.15$, shock 3 : $\delta = 0.9, \xi = 0.85$ shock 4 : $\delta = 0.9, \xi = 1.15$

Learning the Equilibrium

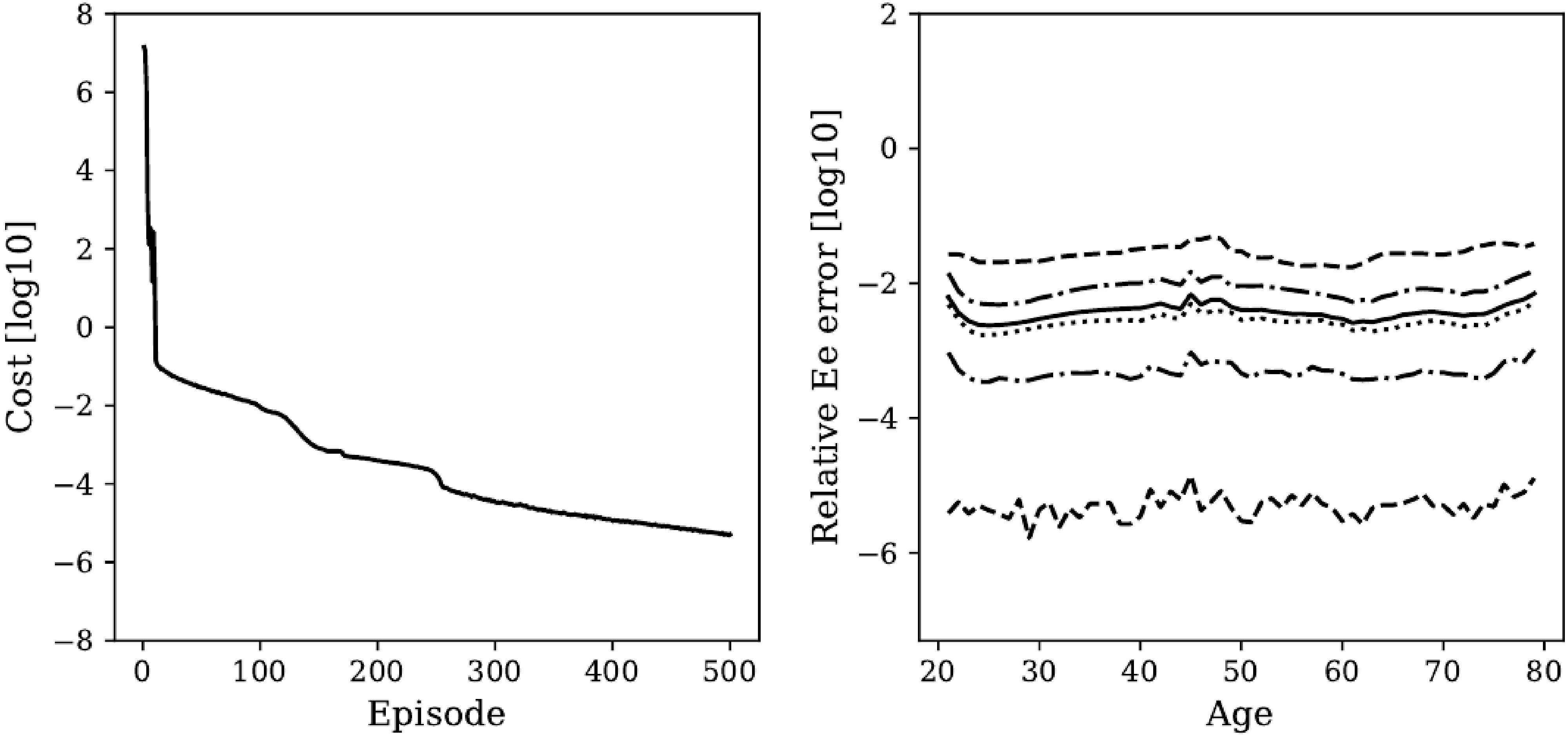


Learning the Equilibrium

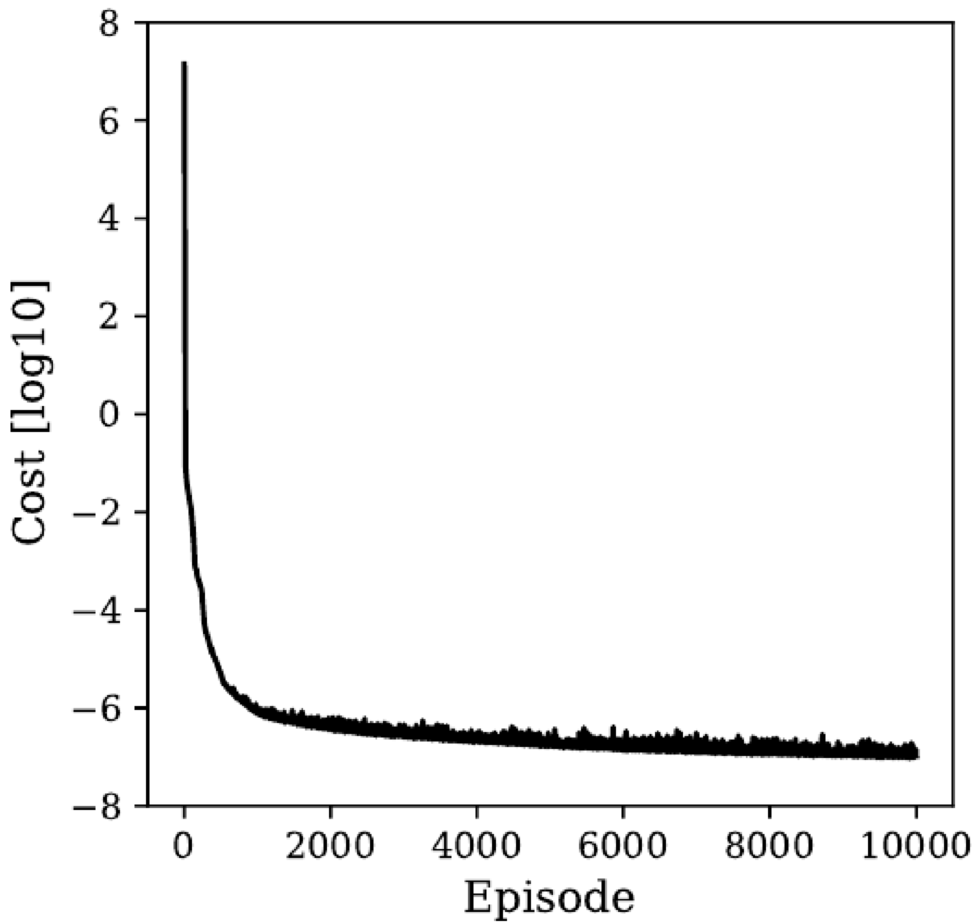


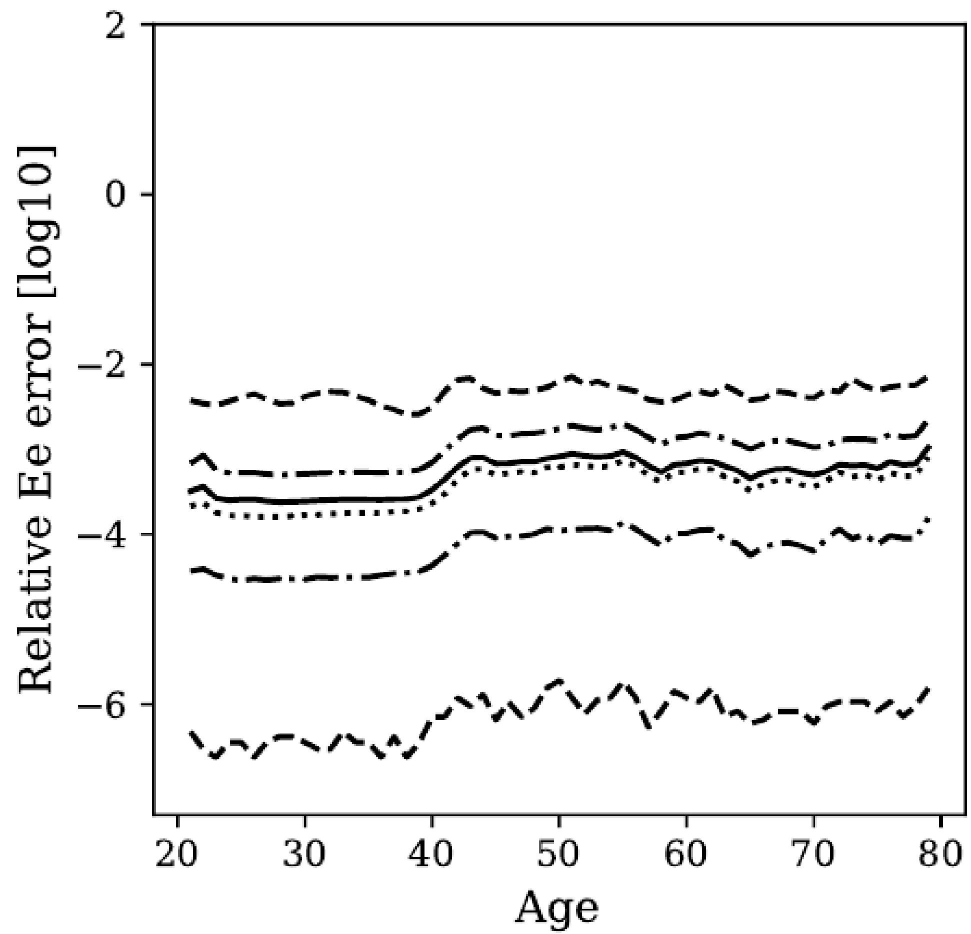


Learning the Equilibrium



Learning the Equilibrium

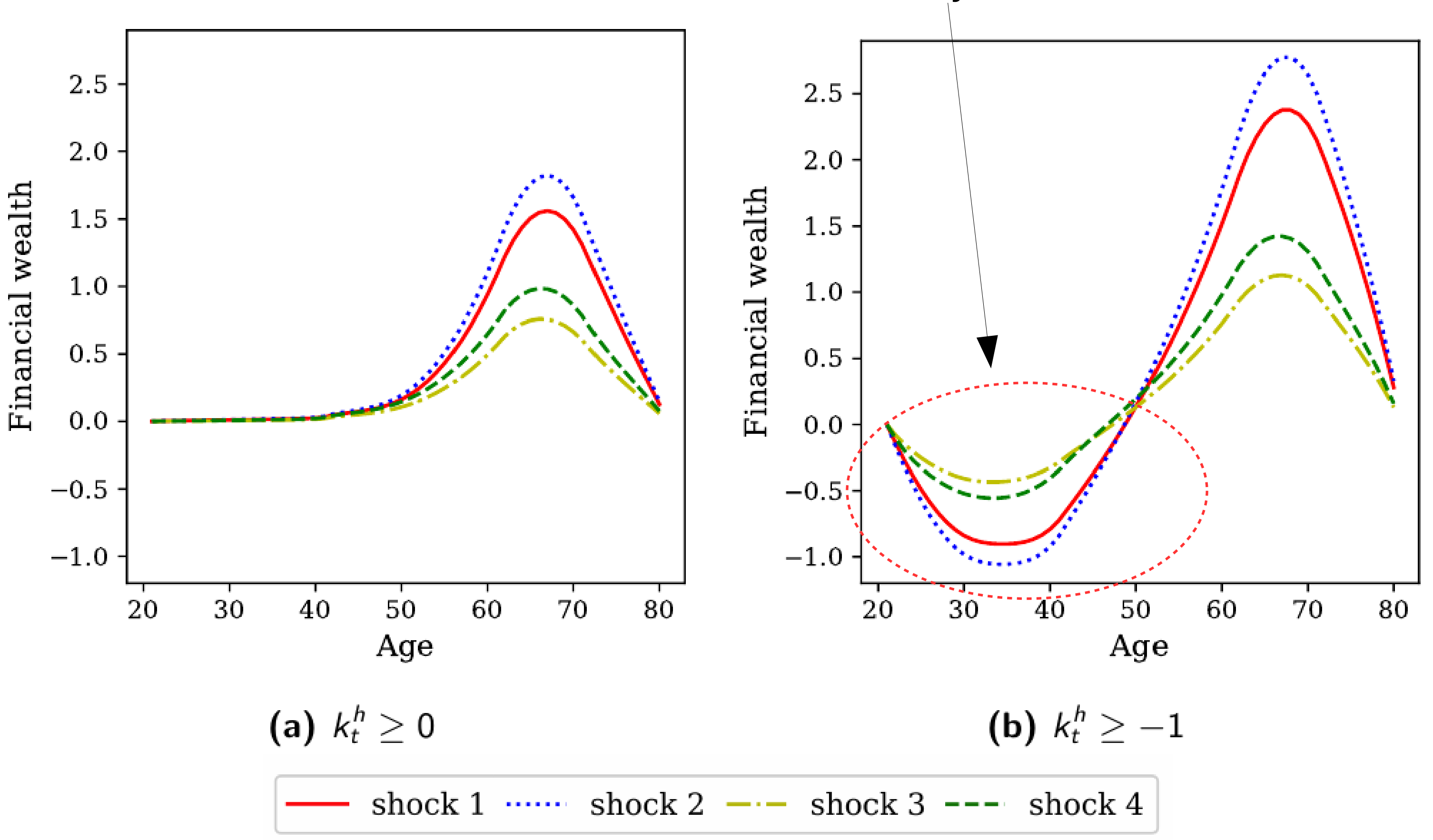




Loosening the borrowing constraint

Financial wealth = value of the capital saved in the last period after returns realized.

Young agents take up debt, pay back later as their labour endowment increases in their later years.



shock $1 : \delta = 0.5, \xi = 0.85$, shock $2 : \delta = 0.5, \xi = 1.15$, shock $3 : \delta = 0.9, \xi = 0.85$ shock $4 : \delta = 0.9, \xi = 1.15$

“To pull a bigger wagon, it is easier to add more oxen than to grow a gigantic ox”

(Skjellum et al. 1999)

Horovod

https://eng.uber.com/horovod/

- We use Horovod to parallelize DEQ.

- Based on MPI.

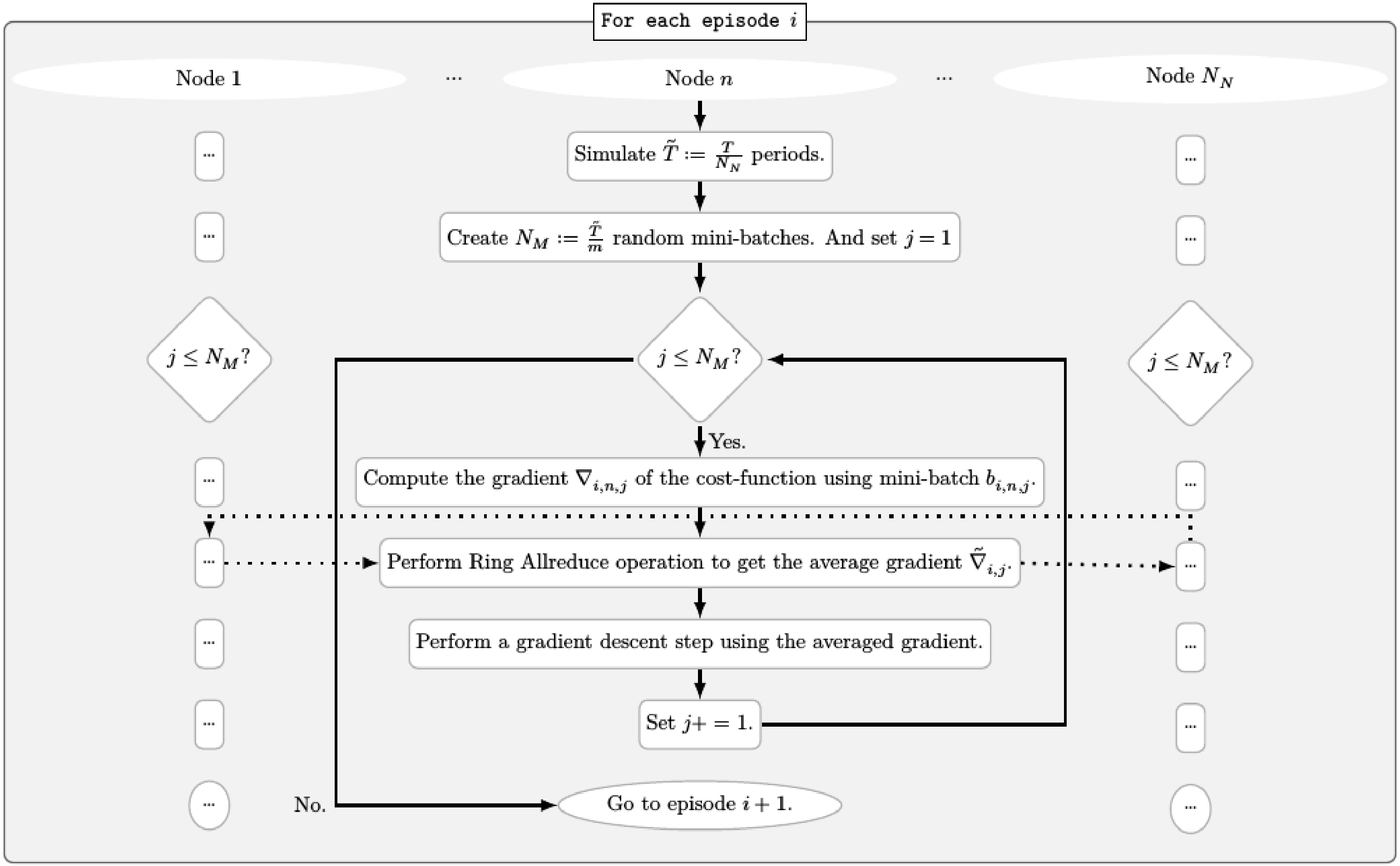
- Idea: **use data-parallelism**.

→ The **generation of training data** is **divided across compute nodes**.

→ Each node computes the gradient of the cost-function concerning the parameters of the neural network on it's given batch of data.

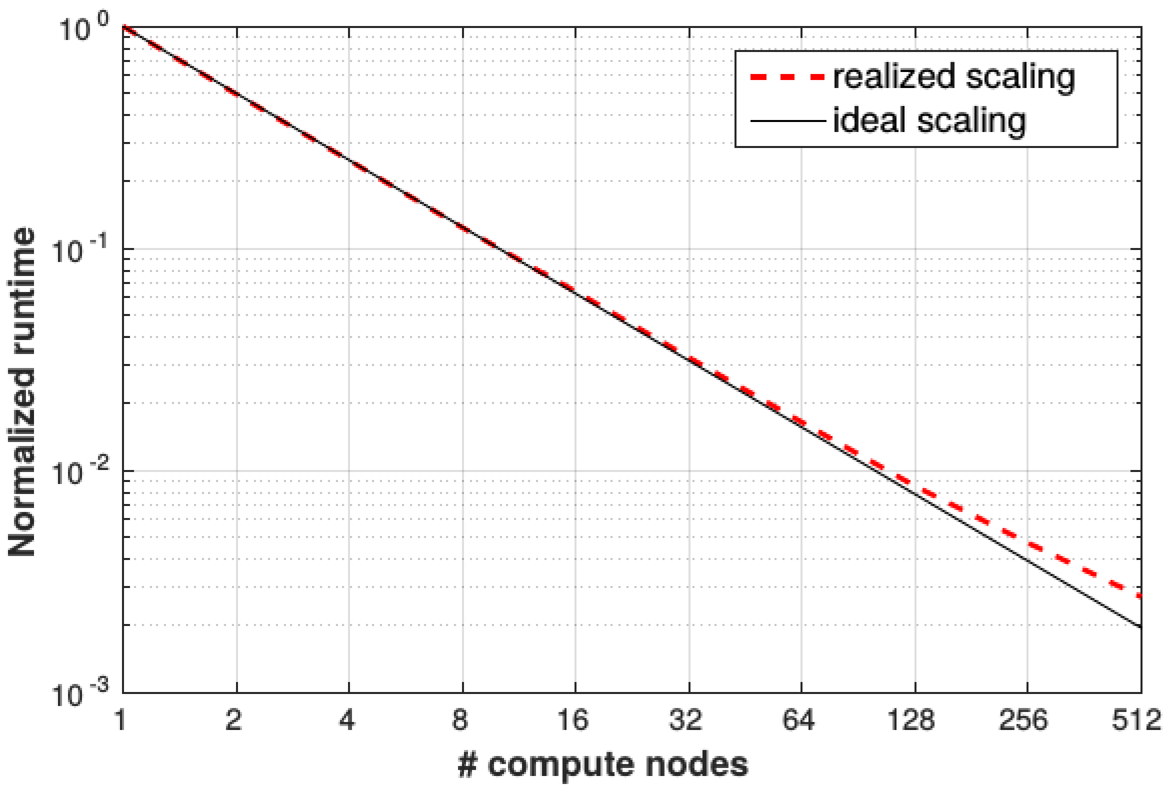
→ Then, all nodes are synchronized and the gradient descent step is taken using the **average gradient**.

Parallelization Scheme



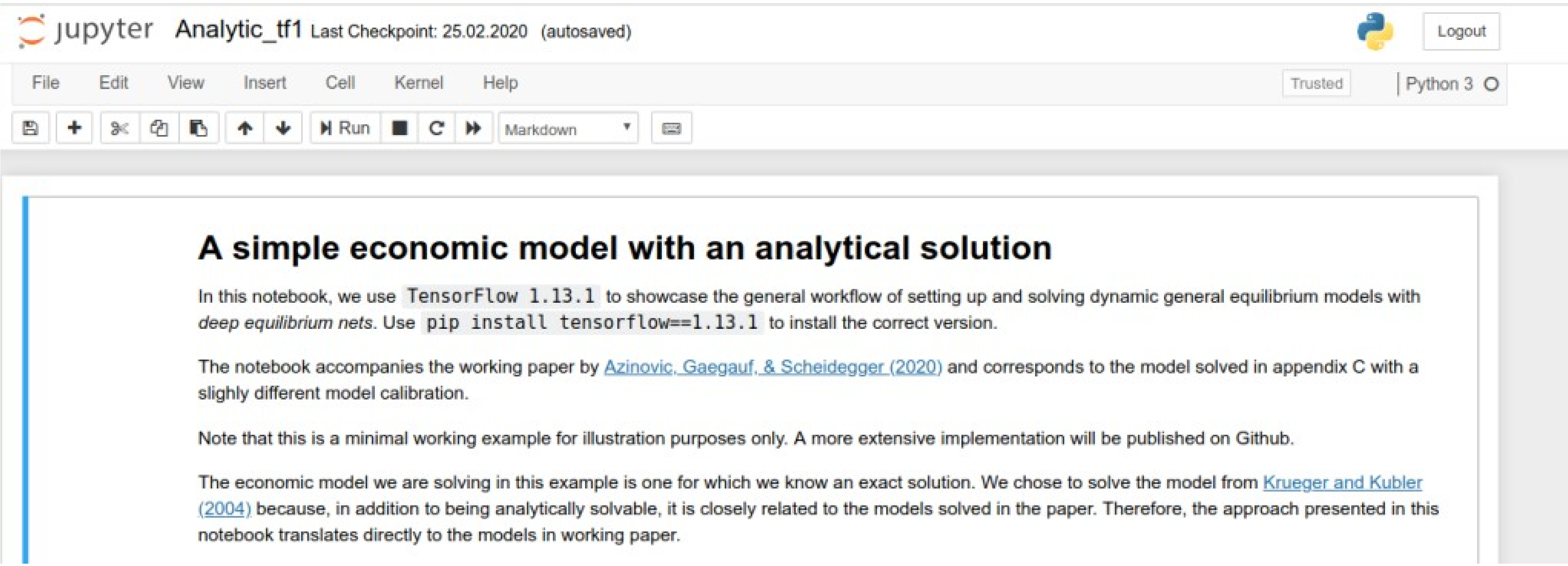
Scalability on “Piz Daint” (CSCS)

- Excellent strong scaling efficiency (over 70% at 512 nodes).



Jupyter Notebook available

Download it here: https://github.com/sischei/DeepEquilibriumNets



A simple economic model with an analytical solution

In this notebook, we use `TensorFlow 1.13.1` to showcase the general workflow of setting up and solving dynamic general equilibrium models with *deep equilibrium nets*. Use `pip install tensorflow==1.13.1` to install the correct version.

The notebook accompanies the working paper by Azinovic, Gaegauf, & Scheidegger (2020) and corresponds to the model solved in appendix C with a slighly different model calibration.

Note that this is a minimal working example for illustration purposes only. A more extensive implementation will be published on Github.

The economic model we are solving in this example is one for which we know an exact solution. We chose to solve the model from Krueger and Kubler (2004) because, in addition to being analytically solvable, it is closely related to the models solved in the paper. Therefore, the approach presented in this notebook translates directly to the models in working paper.

Summary

- Deep learning based, **grid-free, global solution method** to compute approximate recursive equilibria for discrete-time dynamic stochastic economic models with very high-dimensional state spaces.

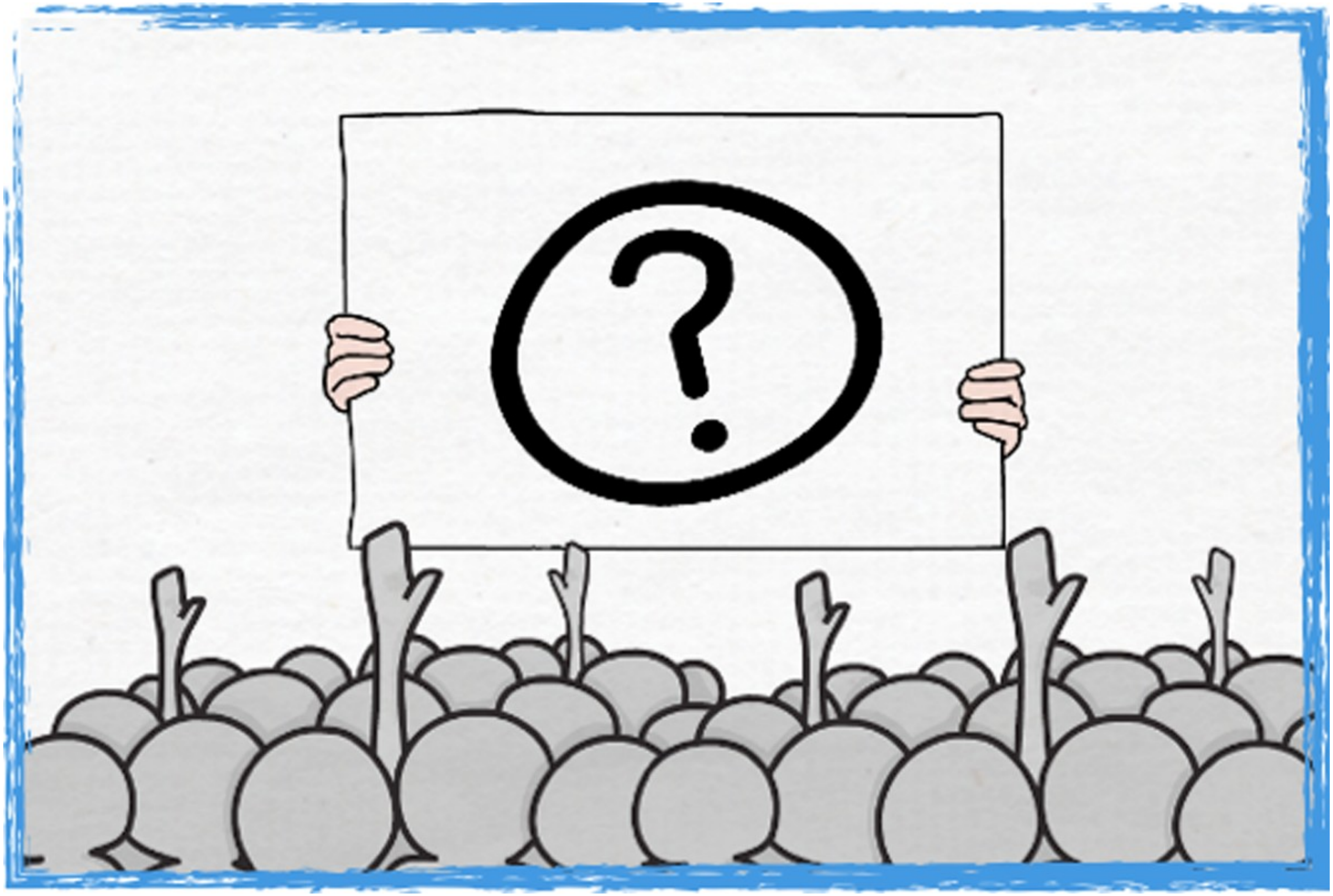
- **Key innovation: use the implied error in the optimality conditions as loss function → training data can be generated at virtually zero cost**.

- We solve for approximate equilibria in an overlapping generations models with 60 generations, aggregate uncertainty, idiosyncratic risk, and occasionally binding constraints, and one-period bonds.

- We obtain average relative errors in the Euler equations of the order $\sim 10^{-4}$.

→ **Generic, scalable and flexible method to address very rich models.**

Questions



A.I.: Example with an exact solution

Taken from Krueger and Kubler (2004), Huffman (1987).

Household side:

$$\begin{cases} \max_{\{c,a\}} \mathrm{E}_t \left[\sum_{i=0}^{N-1} \beta^i \log(c^i_{t+i}) \right] \text{s. t.:} \\ c^h_t + a^h_t = r_t k^h_t + l^h_t w_t \\ k^{h+1}_{t+1} = a^h_t \\ a^{N-1}_{t+N-1} \geq 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} u'(c^h_t) = \mathrm{E}_t \left[r_t u'(c^{h+1}_{t+1}) \right] \\ c^h_t + a^h_t = r_t k^h_t + l^h_t w_t \\ k^{h+1}_{t+1} = a^h_t \\ a^{N-1}_{t+N-1} = 0 \end{cases}$$

Firm side:

$$F(z, K, L) = \xi(z) K^\alpha L^{1-\alpha} + (1 - \delta(z))K$$

$$\max_{\{K,L\}} F(z, K, L) - wL - rK$$

$$\Leftrightarrow \begin{cases} r = \alpha \xi(z) K^{\alpha-1} L^{1-\alpha} + (1 - \delta(z)) \\ w = (1 - \alpha)\xi(z) K^\alpha L^{-\alpha} \end{cases}$$

Calibration:

Number age-groups N	Discount factor β	Relative risk aversion γ	Capital share α
6	0.7	1	0.3

TFP η	Depreciation δ	Persistence TFP $P(\eta_{t+1} = 1.05 \mid \eta_t = 1.05)$ $P(\eta_{t+1} = 0.95 \mid \eta_t = 0.95)$	Persistence depreciation $P(\delta_{t+1} = 0.5 \mid \delta_t = 0.5)$ $P(\delta_{t+1} = 0.9 \mid \delta_t = 0.9)$
$\{0.95, 1.05\}$	$\{0.5, 0.9\}$	0.5 0.5	0.5 0.5

$l^0 = 1$, $l^1 = \cdots = l^5 = 0$.

Exact solution:
save fixed (age dependent) fraction of wealth:

$$a^0_t = \beta \frac{1 - \beta^5}{1 - \beta^6} w_t$$

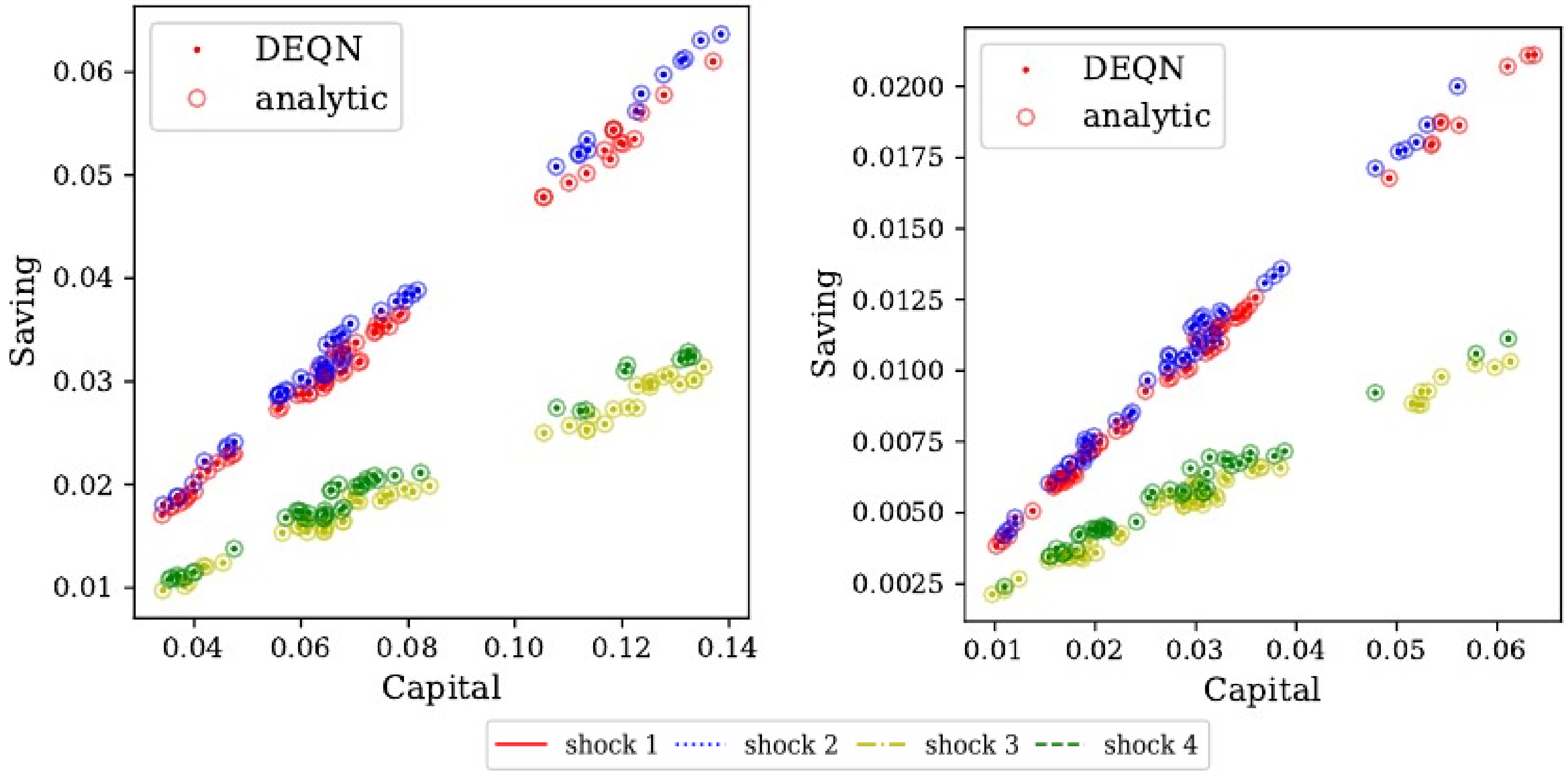
$$a^1_t = \beta \frac{1 - \beta^4}{1 - \beta^5} k^1_t r_t$$

$$\cdots$$

$$a^4_t = \beta \frac{1 - \beta}{1 - \beta^2} k^4_t r_t$$

$$a^5_t = 0$$

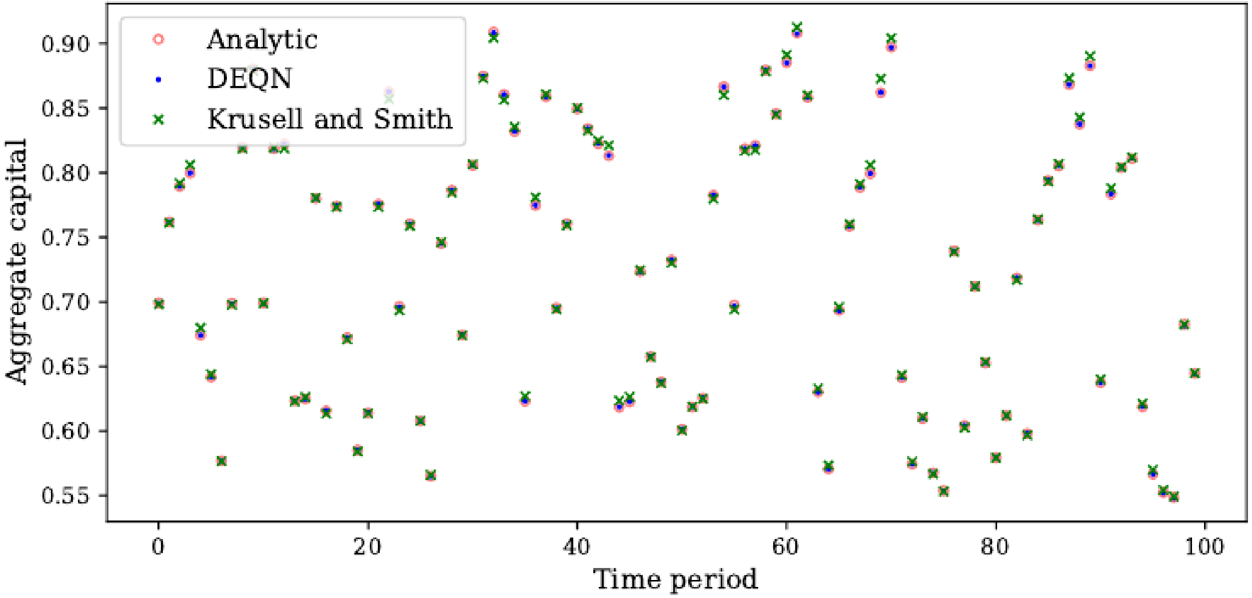
A.I.: Exact and approximated saving decisions vs. capital holding



- **Exact** and **approximated saving decisions** plotted against capital holding in the beginning of the period for 200 simulated states for each of the age-groups 4 and 5.

- **Circles**: the exact solution.
- **Stars**: solutions learned by the deep equilibrium net.

A.I.: Approximate aggregation



	mean	max
DEQN [%]	0.019	0.13
Log-linear forecast [%]	0.24	1.3
DEQN [log10]	-3.72	-2.88
Log-linear forecast [log10]	-2.62	-1.87

- Following Den Haan (2010) and Winberry (2018), we also analyze to which extent approximate aggregation obtains by simulating the linear forecast of aggregate capital forward without updating the value of aggregate capital.

- We do the same using the solution learned by the deep equilibrium net and compare the sequences of aggregate capital on 15, 000 simulated states.