WILL ARTIFICIAL INTELLIGENCE REPLACE COMPUTATIONAL ECONOMISTS ANY TIME SOON?

Lilia Maliar, Serguei Maliar and Pablo Winant

Sync/BondsAll | !SR_Bonds - Ju | !SR_Bonds - Sh | !Training - Jupy | Comparison-B | Bonds_Hank - | Google Drive | model.py - Jup | Show Results - | Bolsa Madrid | Will artificial in

file:///D:/Google Drive New/Deep_Learning_SL/ASSA2021/PresentationASSA2021.html#/2

INTRODUCTION

ESP 6:17 PM 12/23/2020

ARTIFICIAL INTELLIGENCE VERSUS ECONOMIC AGENT

- AI is a device that:
  - has a goal
  - perceives its environment
  - takes actions to achieve its goal
- This looks exactly like an economic agent!
  - goal = utility maximization, profit maximization, etc.
  - environment = state variables
  - action = controls
- So, maybe, general-purpose AI technology can solve the economic models for us

APPLICATIONS: AI VERSUS ECONOMICS

- AI has many impressive applications:
  - recognition of handwriting and speech
  - composing Chopin music, playing Go
  - facilitating computer vision
- At the same time, economists cannot solve:
  - heterogeneous-agent and life-cycle models
  - dynamic games and IO models
  - expensive nonlinear estimation
- So, maybe, machines can solve for us those difficult problems that we want but cannot currently solve

APPROACH IN THIS PAPER

- We do not invent a new computational method for economic models but reformulate the models themselves to fit into the existing state-of-the-art computational methods
- Specifically, we adapt deep learning (DL) framework built on multilayer neural networks with stochastic optimization to solving dynamic economic models
- Our code is written using Google TensorFlow platform - the same software that leads to break-ground applications in data science

THREE MAIN RESULTS

1. We offer a unified approach for casting three fundamental objects of economic dynamics -**lifetime reward, Bellman equation, Euler equation**- into DL objective functions
2. We show how to solve economic models on **random grids** with few grid points (instead of conventional fixed grids with a large set of points)
3. We introduce **"all-in-one expectation operator"** that merges all randomness into a single operator and allows to simultaneously implement both approximation and integration steps

THE APPLICATIONS

1. Consumption-saving problem with borrowing constraints via Fisher-Burmeister function (code is available at QuantEcon.org)
2. Krusell and Smith (1998) model with 1,000 agents: we do not use moments of distributions but construct decision functions of 2,001 state variables, literally feeding the entire distributions into neural networks
3. Large-scale new Keynesian model of the Bank of Canada (in another paper)

MODEL REDUCTION AND ILL-CONDITIONING

- Neural network automatically performs *model reduction!*
  - It learns to extract information from many inputs and condenses it into a smaller set of hidden layers
- Moreover, neural network can deal with *ill conditioning*
  - It learns to ignore redundant and collinear variables
- Taken together, these features allow us to deal with a huge state space of 2,001 variables

RELATION TO COMPUTATIONAL ECONOMICS

- The lifetime-reward maximization method is related to an indirect inference procedure of Smith (1987)
- The Euler-equation method is related to seminal contributions to numerical solution methods in economics: the projection method of Judd (1992) and PEA of Den Haan and Marcet (1990)
- The Bellman-equation method is related to conventional value and policy function iteration (see e.g., Rust, 1996, Judd, 1998, Santos, 1999, Aruoba et al., 2006, Stachurski, 2009)

RELATION TO THE LEARNING LITERATURE

- *Supervised Learning*: Duffy and McNelis (2001), Duarte (2018), Fernandez-Villaverde et al. (2018), Villa and Valaitis (2019), Lepetyuk et al. (2019), Azinovic et al. (2019)
- *Unsupervised Learning*: Judd et al. (2011), Maliar and Maliar (2015), Coleman et al. (2018)
- *Reinforcement Learning*: Sutton and Barto (2018), Powell (2010), Lepetyuk and Jirniy (2012)

We differ in that we cast the entire economic model into DL objective function

PLAN OF THE TALK

- Canonical supervised learning
- Deep learning and neural networks
- Stochastic optimization
- Deep learning algorithm for optimization
- Casting economic models into DL objective functions
- Consumption-saving model
- Krusell and Smith (2019) model
- Conclusion

CANONICAL SUPERVISED LEARNING

CANONICAL SUPERVISED LEARNING = JUST A REGRESSION

- Output data: $y_i \in \mathbb{R}^{d_y}$
- Input data (features): $x_i \in \mathbb{R}^{d_x}$
- Approximation (prediction) function: $\varphi : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$
- *Goal:* to learn φ s.t. given $x_i \in \mathbb{R}^{d_x}$, the value $\varphi(x_i)$ provides an accurate prediction about true y_i
- The function φ is selected within a given family of parametric functions $\{\varphi(\cdot;\theta) : \theta \in \mathbb{R}^{d_\theta}\}$, θ = parameters vector.

EXPECTED LOSS

- *Loss function* $\ell : \mathbb{R}^{d_y} \longrightarrow \mathbb{R}$ = difference between the true output y and the predicted output $\varphi(x;\theta)$.
- Minimize the *expected loss (expected risk)* $\Xi(\theta)$:

$$\Xi(\theta) \equiv \int_{\mathbb{R}^{d_x} \to \mathbb{R}^{d_y}} \ell(\varphi(x;\theta), y) dP(x,y) =$$

$$= E_x[\ell(\varphi(x;\theta), y)];$$

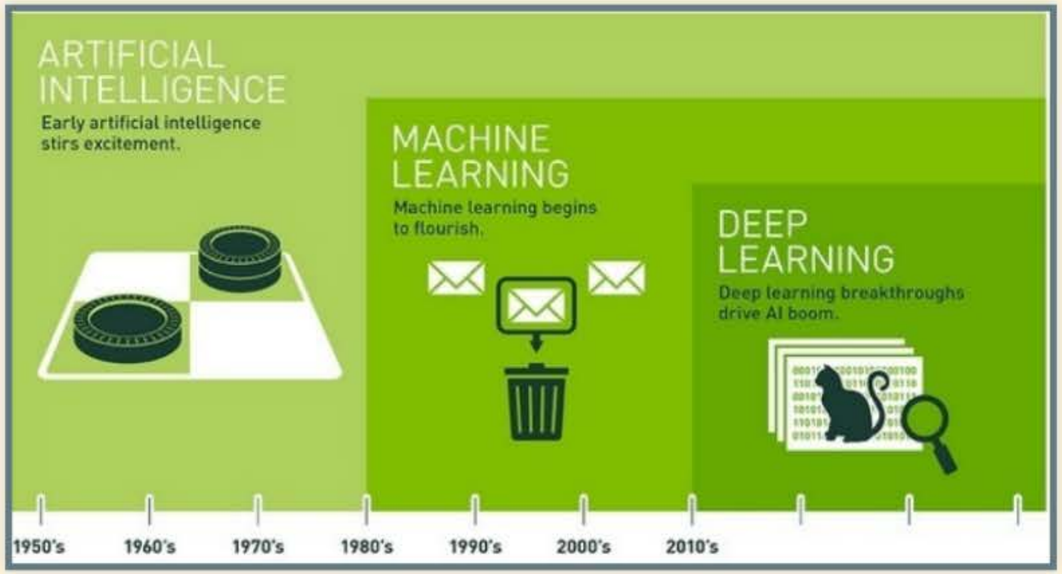
$P : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \longrightarrow [0,1]$ = a joint probability distribution function

EMPIRICAL LOSS

- In practice, learn θ that minimizes *empirical loss*: $\Xi_n(\theta) \equiv \frac{1}{n} \sum^n{}_{i=1} \ell(\varphi(x_i;\theta), y_i)$, where $\{(x_i, y_i)\}_{i=1}^n$ is a given set of input-output pairs
- *Example: linear regression* $y = \theta x$, *where*
  - $\varphi(x;\theta) = \theta x$ *is the prediction function and*
  - $\ell(\theta x, y) = (y - \theta x)^2$ *is the loss function*

BACKGROUND ON DEEP LEARNING

AI, ML AND DL



- Artificial neural networks
  - invented in 50s
  - widely used in 80s and early 90s; less so in 90s
  - recent resurgence: state-of-the-art technique

DEEP NEURAL NETWORKS

- DL models are based on an artificial neural network
- in DL, each level learns to transform its input data into a slightly more abstract and composite representation
- the "deep" in "deep learning" indicates that there are *multiple* layers through which the data are transformed
- endowing neural networks with multiple layers is what precisely led to the break-through in the field of DL

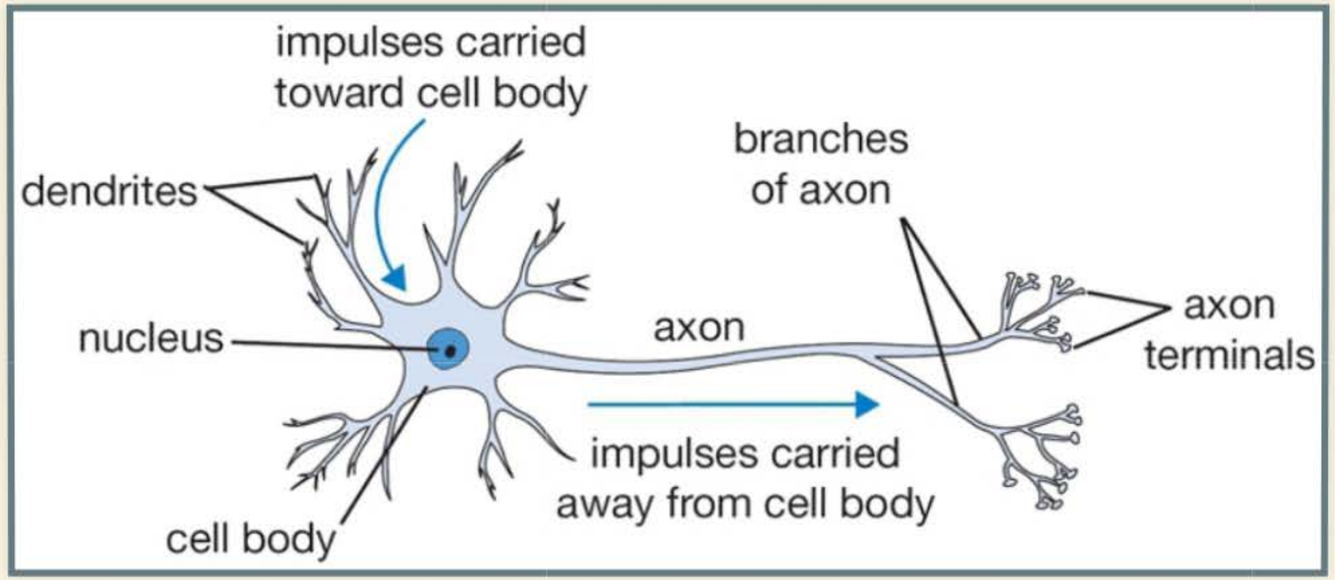
GOOGLE TRENDS

- the word "deep learning" on Google Trends

- software "Alpha-Go" defeated the South Korean champion Lee Se-dol in the game Alpha-Go in March 2016

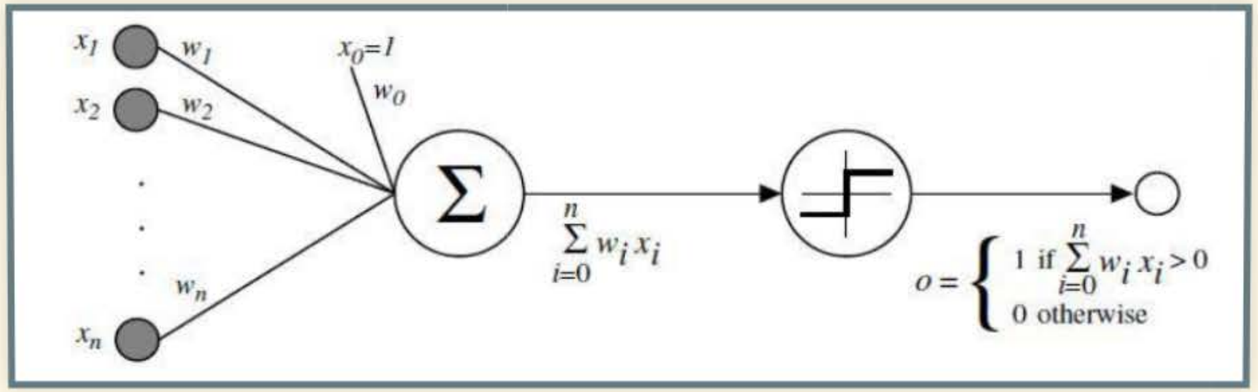
BIOLOGICAL NEURON

- algorithms trying to mimic neurons in the brain



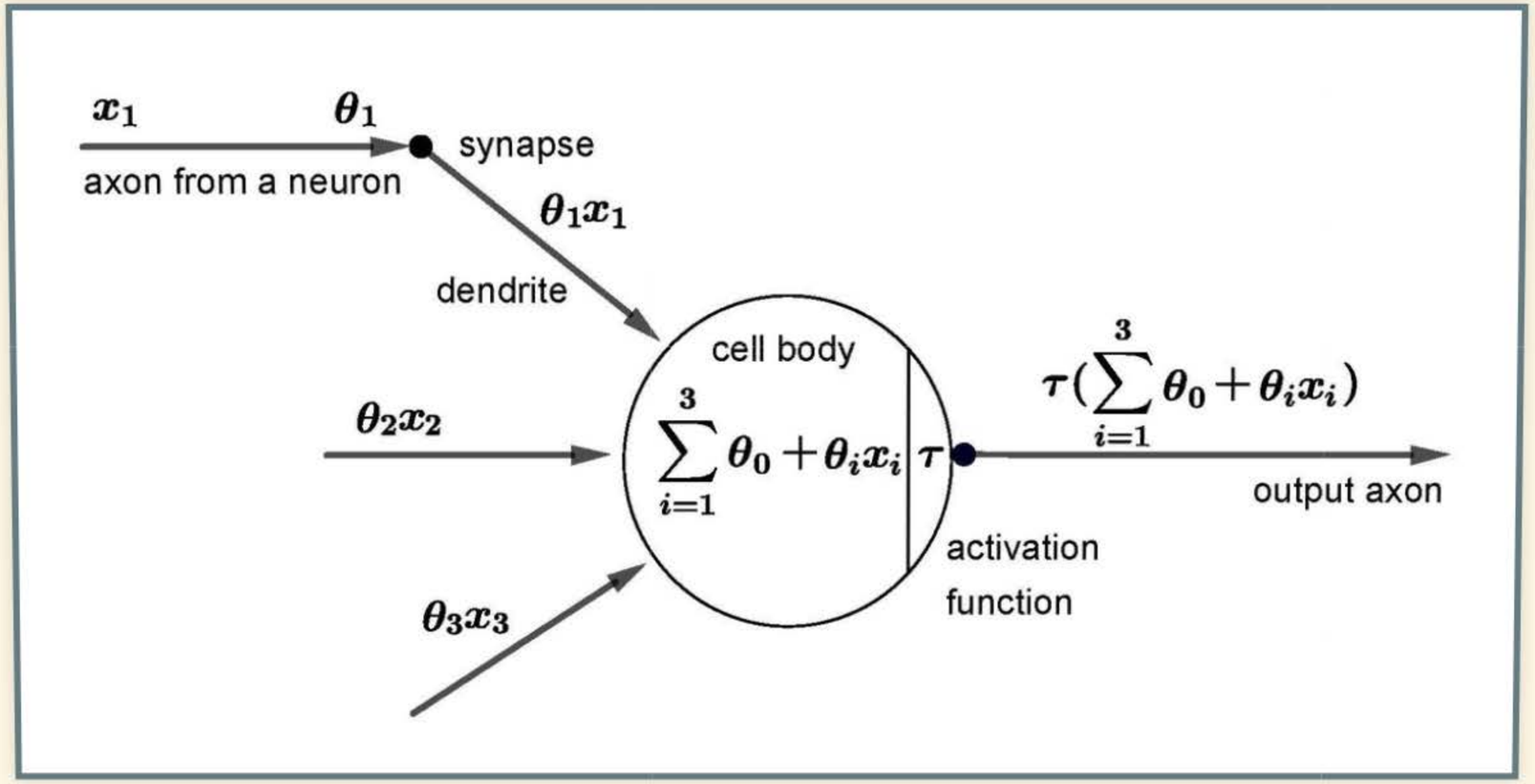
EARLY ARTIFICIAL NEURON

- a perceptron - the first simplified model of a biological neuron
- invented by a psychologist Frank Rosenblatt (1957)



MODERN ARTIFICIAL NEURON

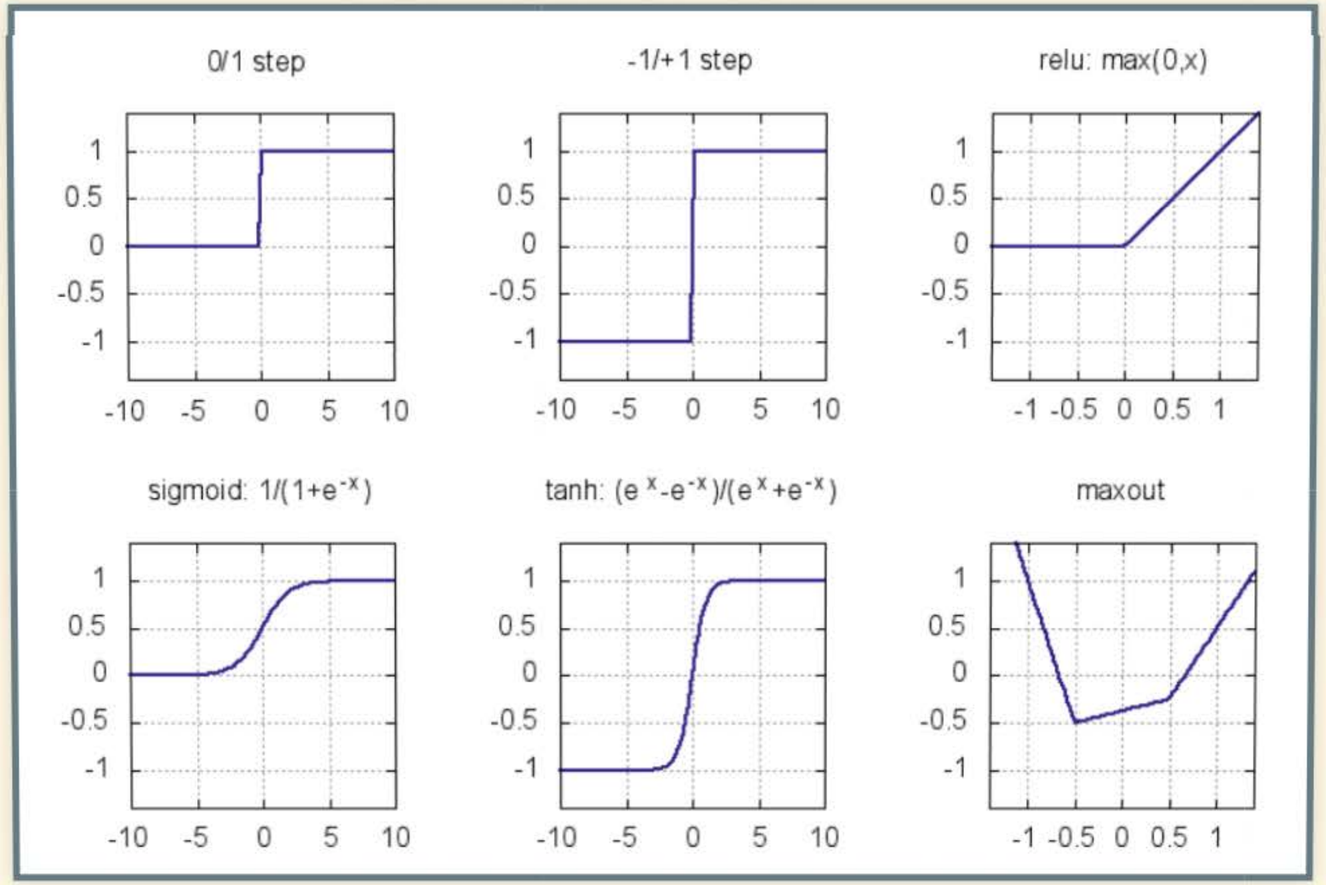
- arbitrary activation function τ



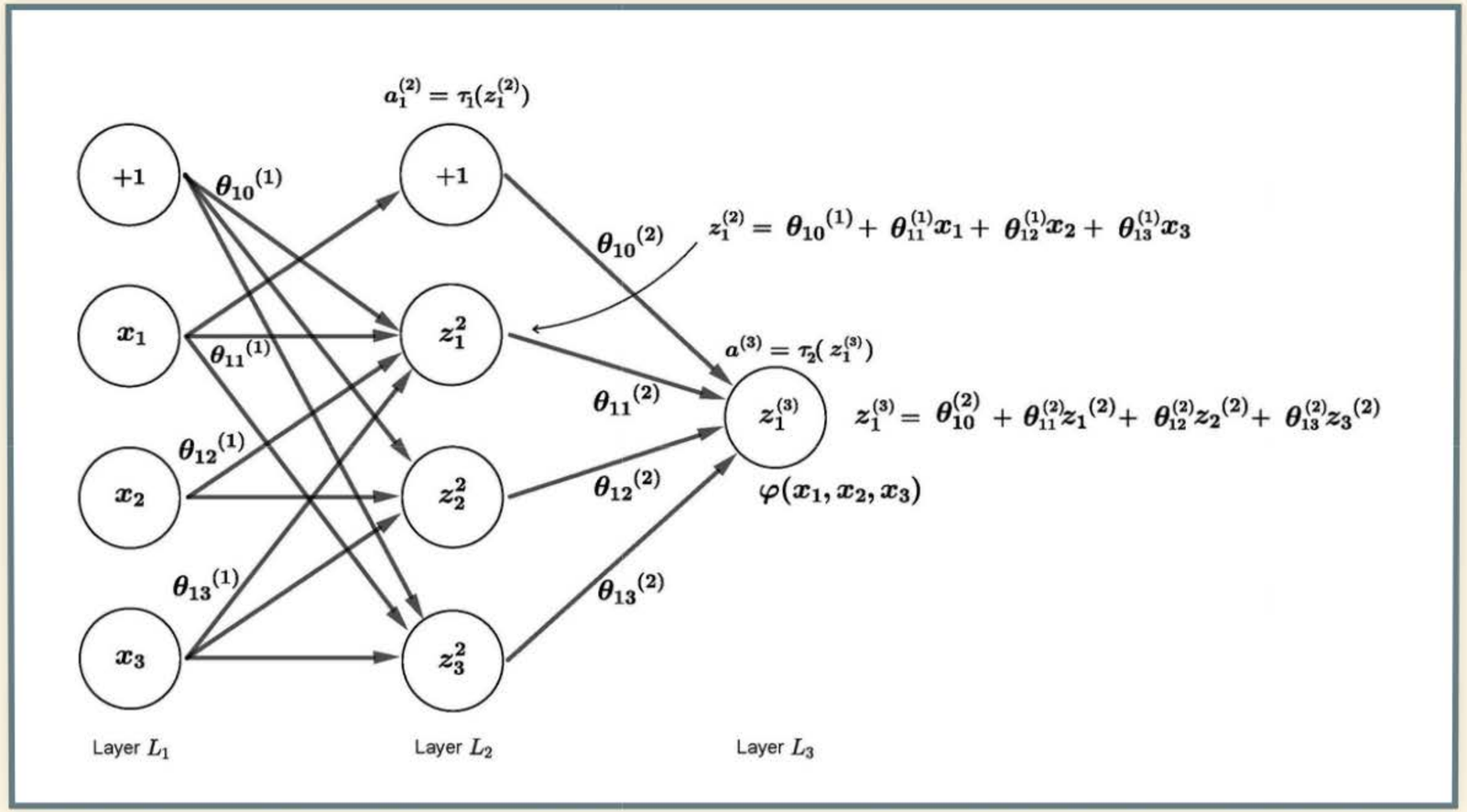
ACTIVATION FUNCTIONS

- heaviside step function: $\tau(x) = 1_{x \geq 0}$
- sigmoid (logistic): $\sigma(x) = \frac{1}{1+e^{-x}}$
  - fact: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
- tanh (hyperbolic tangent): $\tau(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
- relu (rectified linear units): $r(x) = max(0, x)$
- leaky relu: $r(x) = max(\kappa x, x), \kappa \geq 0$
- maxout: $\tau(x) = max(a_1 x + b_1, a_2 x + b_2, a_3 x + b_3)$

ACTIVATION FUNCTIONS



NEURAL NETWORK: EXAMPLE



NEURAL NETWORK: NOTATION

- parameters: $(\theta^{(1)}, \theta^{(2)}) \equiv \theta$;
- $\theta_{ij}^{(l)}$: weight associated with the connection between neuron j in layer l and the neuron i in layer $l+1$, for $i, j = 0, 1, 2, 3$, and $l = 1, 2$;
- $\theta^{(l)}$: matrix made by the weights between the neurons of layer l and the next layer $l+1$;
- $z_i^{(l)}$: nonactivated output of neuron i in layer l;
- τ_l: activation function of layer l;
- $a_i^{(l)}$: activated output of neuron i in layer l.

NEURAL NETWORK: FORWARD PROPAGATION

- input $x = (1, x_1, x_2, x_3), a^{(1)} = x$
- $z_1^{(2)}$: nonactivated output of neuron 1 in layer 2
  - $z_1^{(2)} = \theta_{10}^{(1)} + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3$
- $a_1^{(2)}$: activated output of neuron 1 in layer 2
  - $a_1^{(2)} = \tau_1(z_1^{(2)})$
  - or $a^{(2)} = \tau_1(\theta^{(1)} x)$
- $a^{(3)}$: activated output in layer 3
  - $a^{(3)} = \tau_2(z_1^{(3)})$
  - or $a^{(3)} = \tau_2(\theta^{(2)} a^{(2)})$
- $a^{(3)} = \varphi(x; \theta)$: predicted output

BACKGROUND ON TRAINING OF MACHINES

TRAINING NEURAL NETWORKS

- Training of the machine = find the parameters of the approximating function
- $\Xi(\theta) = E_\omega[\xi(\omega;\theta)]$, and $\Xi_n(\theta) = \frac{1}{n}\sum_{i=1}^{n}\xi(\omega_i;\theta)$, $\{\omega_i\}_{i=1}^n$ = the given set of input-output pairs $\{(x_i, y_i)\}_{i=1}^n$
- define empirical risk $\Xi(\theta)$ by MSE: $\Xi_n(\theta) = \frac{1}{n}\sum^{n}{}_{i=1}(\varphi(x;\theta) - y)^2$
- $\Xi_n(\theta)$ allows us to evaluate how different $\varphi(x;\theta)$ is from true y
- need to compute the gradient $\nabla_\theta \Xi_n$

BACKPROPAGATION: IDEA

- To compute the gradient, use backpropagation:
- perform forward propagation to calculate all the parameters θ
- calculate an error term backward for each neuron, starting from the output layer towards the input layer
- assess how much each neuron can be held responsible for a potential error in the network's output

STOCHASTIC GRADIENT

- *Gradient Descent*: (learning rate λ)

$$\theta_{k+1} \leftarrow \theta_k - \lambda_k \nabla_\theta E_\omega \left[\xi(\omega; \theta_k)\right]$$

- *Stochastic Gradient Descent*:

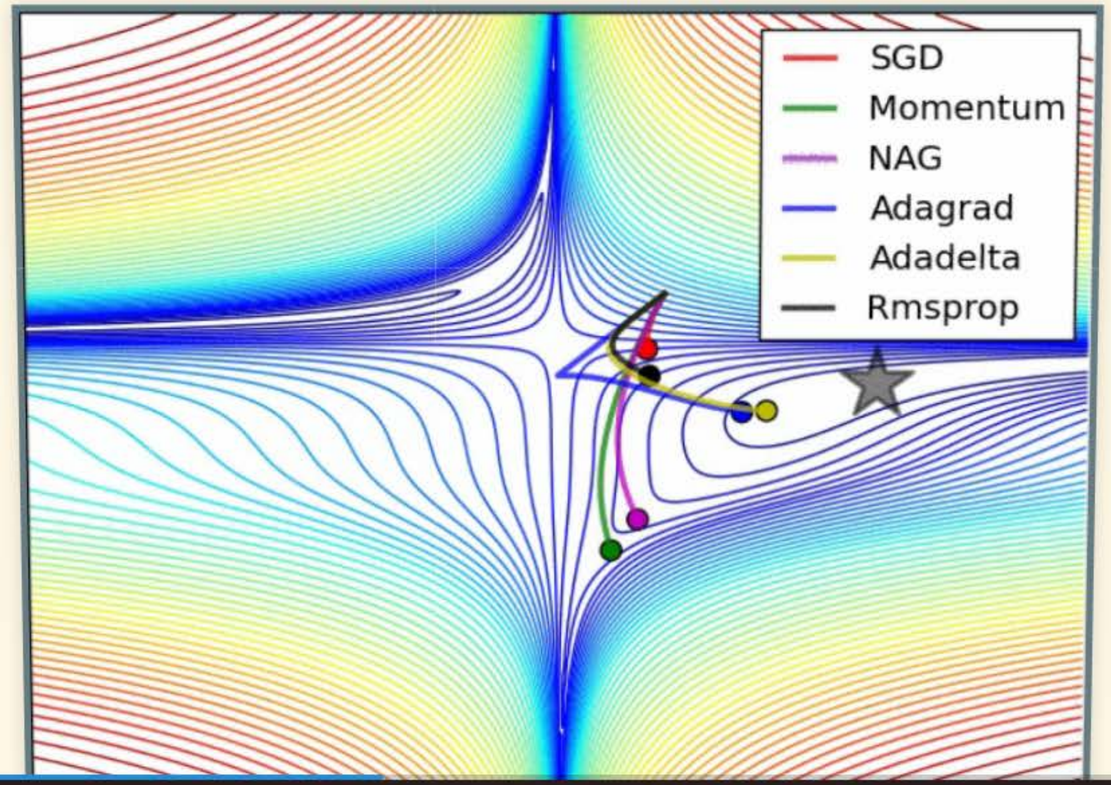
$$\theta_{k+1} \leftarrow \theta_k - \lambda_k \nabla_\theta \xi(\omega; \theta_k)$$

- *Batch Stochastic Gradient Descent*:

$$\theta_{k+1} \leftarrow \theta_k - \frac{\lambda_k}{n} \sum_{i=1}^{n} \nabla_\theta \xi(\omega_i; \theta_k)$$

GRADIENT DESCENT

- Goal: minimize theoretical risk $\Xi(\theta) = E_\omega \left[\xi(\omega; \theta)\right]$
- *Gradient Descent*: $\theta_{k+1} \leftarrow \theta_k - \lambda_k \nabla_\theta E_\omega \left[\xi(\omega; \theta_k)\right]$



A DEEP LEARNING ALGORITHM FOR OPTIMIZATION

DECISION FUNCTIONS ARE IMPLICIT IN ECONOMICS

- Canonical supervised learning is designed to estimate a regression $y = \varphi(x; \theta)$
- It is not directly suited for solving economic models where we need to find value/policy functions $\varphi(x; \theta)$ because the value of y is unknown
- For example, if we approximate the capital policy function $k' = \varphi(k; \theta)$, we don't know what k' is
- Therefore, we generalize canonical supervised learning to the case when y is defined implicitly

GENERALIZATION OF CANONICAL SUPERVISED LEARNING

- Introduce new variable $\omega \equiv (x, y)$
- Then, loss function is $\xi(\omega; \theta) \equiv \ell(\varphi(x; \theta), y)$
- This change of variables converts supervised learning into a generic optimization of expectation functions
- It also provides a link to unsupervised learning!

AI ALGORITHM FOR OPTIMIZATION

1. Define theoretical and empirical risks
   $\Xi(\theta) = E_\omega[\xi(\omega;\theta)]$, and $\Xi_n(\theta) = \frac{1}{n}\sum_{i=1}^n \xi(\omega_i;\theta)$,
   $\{\omega_i\}_{i=1}^n$ = the given set of features (data points).
2. Define a family of parametric functions
   $\{\varphi(\cdot;\theta) : \theta \in \mathbb{R}^{d_\theta}\}$, θ = parameters vector.
3. Train the machine, i.e., find θ that minimizes the empirical risk, typically using stochastic gradient or batch stochastic gradient methods.
4. Check accuracy of the constructed approximation.

We are just left to frame the economic model as an objective for the empirical risk!

CONSUMPTION-SAVING PROBLEM

OBJECTIVE FUNCTIONS FOR DL

Three fundamental objects of economic dynamics:

- lifetime reward of a consumer (to be maximized)
- residuals in Bellman equation (to be minimized)
- residuals in Euler equation (to be minimized)

We show how to convert these three objects into the objective functions of DL algorithm

CONSUMPTION-SAVING PROBLEM WITH MULTIPLE SHOCKS

$$\max_{c_t, w_{t+1}} E_0 \left[\sum_{t=0}^{\infty} \beta^t e^{\chi_t} u(c_t) \right]$$

$$\text{s.t. } w_{t+1} = r e^{\varrho_t} (w_t - c_t) + e^{y_t} e^{p_t},$$

$$c_t \leq w_t,$$

$$(z_0, w_0) \text{ given.}$$

- c_t = consumption; w_t = cash-on-hand; $r \in (0, \frac{1}{\beta})$.
- each $z_t \in \{y_t, p_t, \varrho_t, \chi_t\}$ follows an AR(1) process:
  $z_{t+1} = \rho z_t + \sigma \epsilon_{t+1}$.

OPTIMALITY CONDITIONS

- *Kuhn-Tucker conditions:* $c - w \leq 0, h \geq 0$ and $(c - w)h = 0, h \equiv u'(c)e^{\chi - \varrho} - \beta r E[u'(c')e^{\chi'}] =$ Lagrange multiplier.
- *Bellman equation:* $V(z, w) = \max_{c, w'} \{u(c) + \beta E_\epsilon [V(z', w')]\}$

CALIBRATION

- $u(c) = \frac{c^{1-\gamma}-1}{1-\gamma}, \gamma = 2$
- $\beta = 0.9, r = 1.04, \sigma_y = 0.1$
- Baseline unishock model: a temporary i.i.d. income shock y
- Multi-shock model: $\rho_y = 0.9$ and $\sigma_y = 0.1$; $\rho_p = 0.999$ and $\sigma_p = 0.001$; $\rho_\varrho = 0.2$ and $\sigma_\varrho = 0.001$; $\rho_\chi = 0.9$ and $\sigma_\chi = 0.01$

COMPUTATIONAL DETAILS

- Our code is in Python and Google's TensorFlow.
- We use Dolo interface written by Pablo Winant.
- *Decision function:*
  - parameterize $\frac{c_t}{w_t} \equiv \zeta_t = \sigma(\zeta_0 + \varphi(z_t, w_t; \theta))$,
  - $\varphi(\cdot)$ = neural network with 2 hidden layers,
  - $\sigma(x) = \frac{1}{1+e^{-x}}$ = sigmoid function,
  - hidden layers include 32 (leaky) relu neurons.
- *Value function:*
  - Parameterize $V_t = V_0 + \varphi(z_t, w_t; \theta)$,
  - $\varphi(\cdot)$ = neural network.

COMPUTATIONAL DETAILS (CONT.)

- *Training*: 16 random draws & $K = 50,000$ iterations using
  - a mini-batch gradient descent (GD) method;
  - ADAM - a version of the stochastic GD algorithm.
- *Accuracy*:
  - $1,024$ random draws;
  - 10-node Gauss-Hermite quadrature rule for the unishock baseline case;
  - 100-node Monte Carlo integration method for the multishock case.

OBJECTIVE 1: LIFETIME REWARD MAXIMIZATION

- Parameterize consumption function $c(\cdot;\theta)$
- For a random vector $\omega \equiv (z_0, w_0, \epsilon_1, \ldots, \epsilon_T)$ construct the expectation operator:

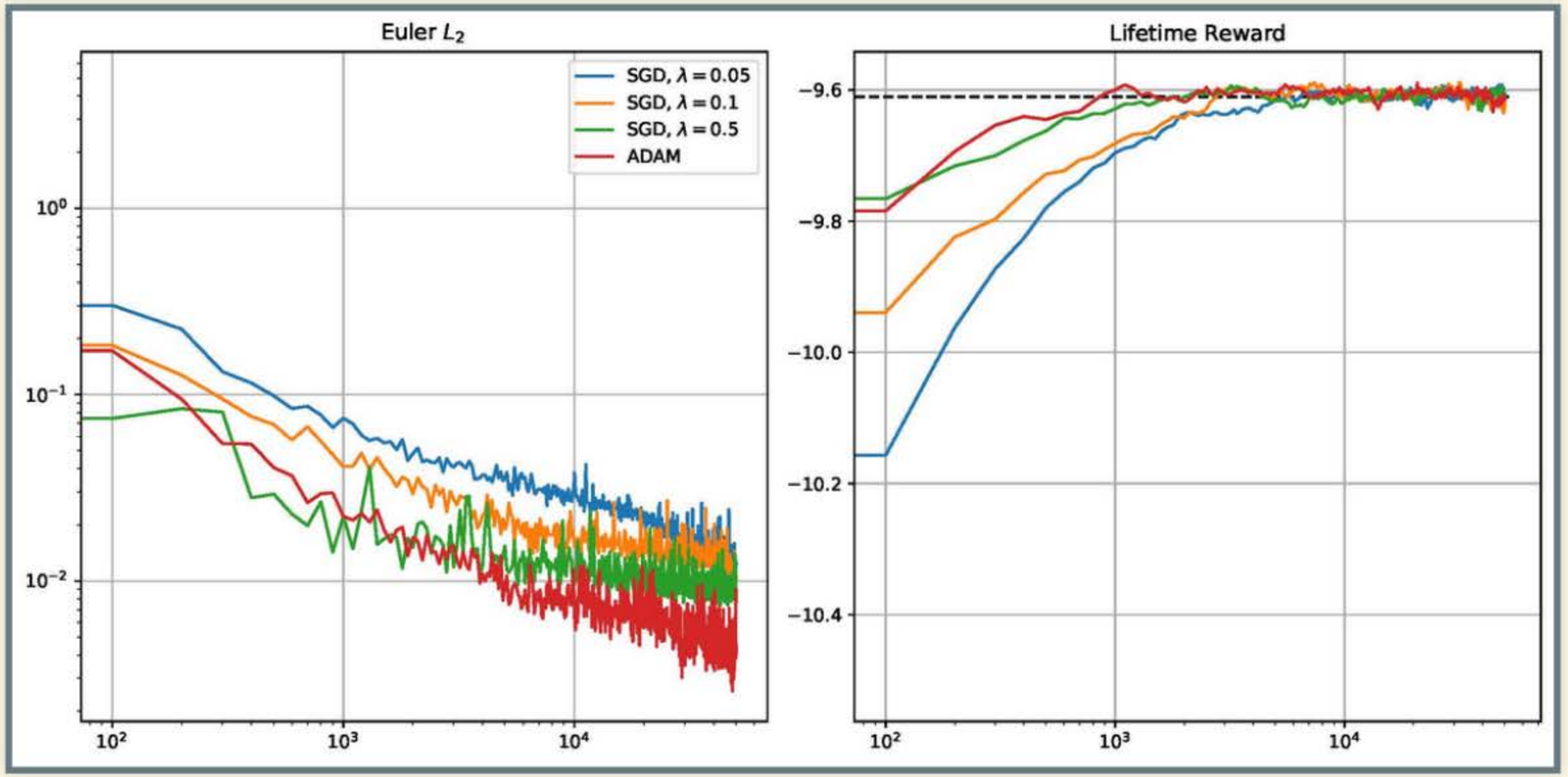
$$\Xi(\theta) \equiv E_\omega[\sum_{t=0}^{T} \beta^t u(c(z_t, w_t; \theta))]$$

- E_ω has two types of randomness: a random state (m_0, s_0) and random future shocks $(\epsilon_1, \ldots, \epsilon_T)$
- We call E_ω *all-in-one expectation operator*: it summarizes all random variables in one place

ALL-IN-ONE EXPECTATION OPERATOR

- A remarkable *distributive property* of $\Xi(\theta) = E_{(m_0,s_0,\epsilon_1,\ldots,\epsilon_T)}[\sum_{t=0}^{T} \beta^t u(c(z_t, w_t; \theta))]$
- The same n random draws are used to evaluate all the expectation functions at once
- In contrast, an integration method that constructs 2 expectation functions $E_{(m_0,s_0)}[\cdot]$ and $E_{(\epsilon_1,\ldots,\epsilon_T)}[\cdot]$ separately would require n draws for evaluating 1st expectation and n' draws for evaluating 2nd expectation -- $n \times n'$ draws in total

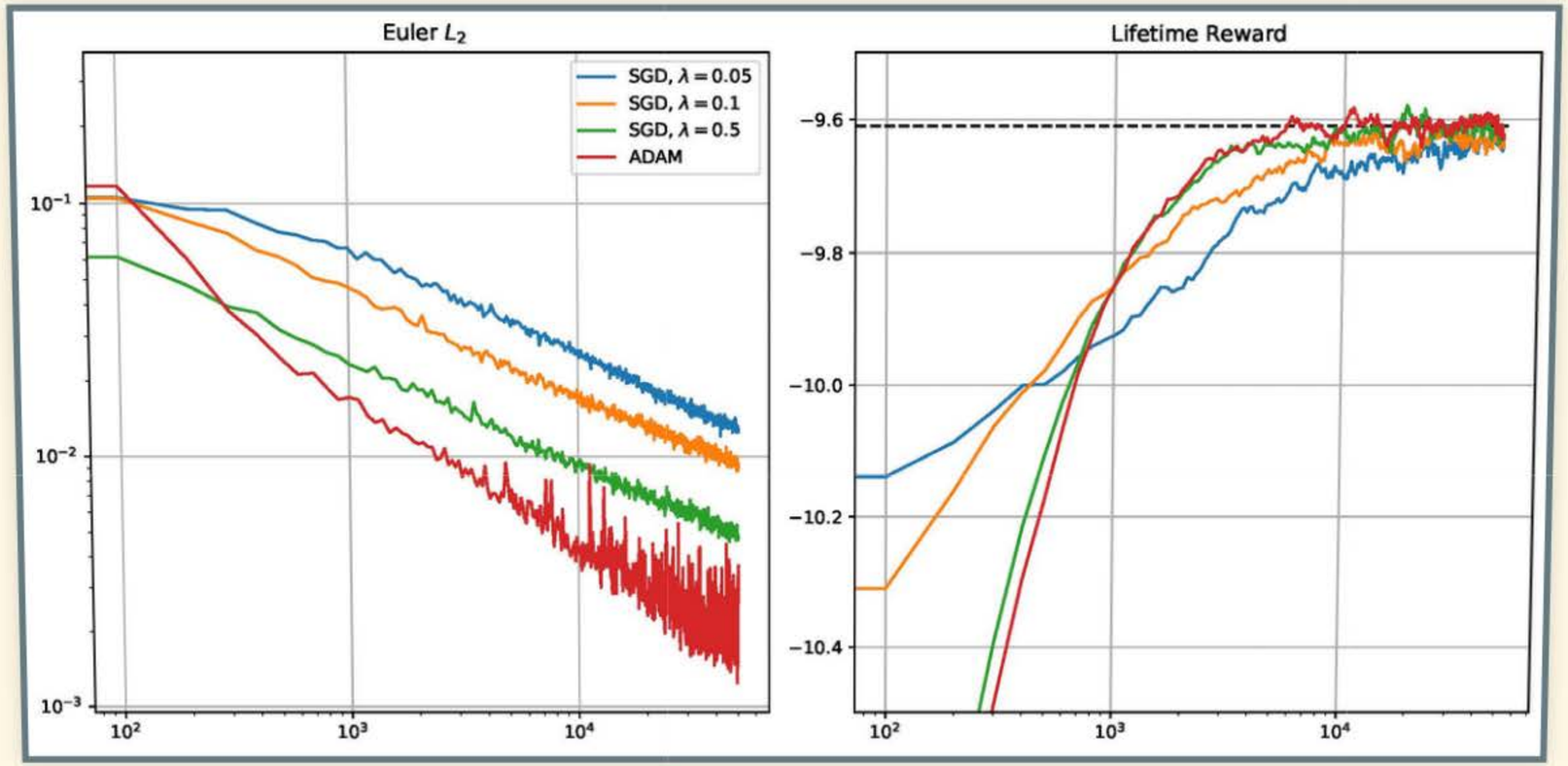
OBJECTIVE 1: LIFETIME REWARD MAXIMIZATION (CONT.)



OBJECTIVE 2: EULER-EQUATION METHOD WITH KUHN-TUCKER CONDITIONS

- *Fischer-Burmeister (FB) function:*
  - $\Psi^{FB}(a,b) = a + b - \sqrt{a^2 + b^2} = 0$
  - with $a = c - w$ and $b = u'(c) - \beta r e^{\varrho_t} E_\epsilon[u'(c')]$
  - represents Kuhn-Tucker conditions $a \geq 0, b \geq 0$ and $ab = 0$ but differentiable
- *Objective function:* For a random vector (z, w) construct the expectation operator $\Xi(\theta) \equiv E_{(z,w)}[\Psi^{FB}(c - w, u'(c) - \beta r e^{\varrho_t} E_\epsilon[u'(c'))])]^2$
- Not all-in-one-expectation operator yet!

OBJECTIVE 2: EULER-RESIDUAL MINIMIZATION



OBJECTIVE 3: BELLMAN-RESIDUAL MINIMIZATION

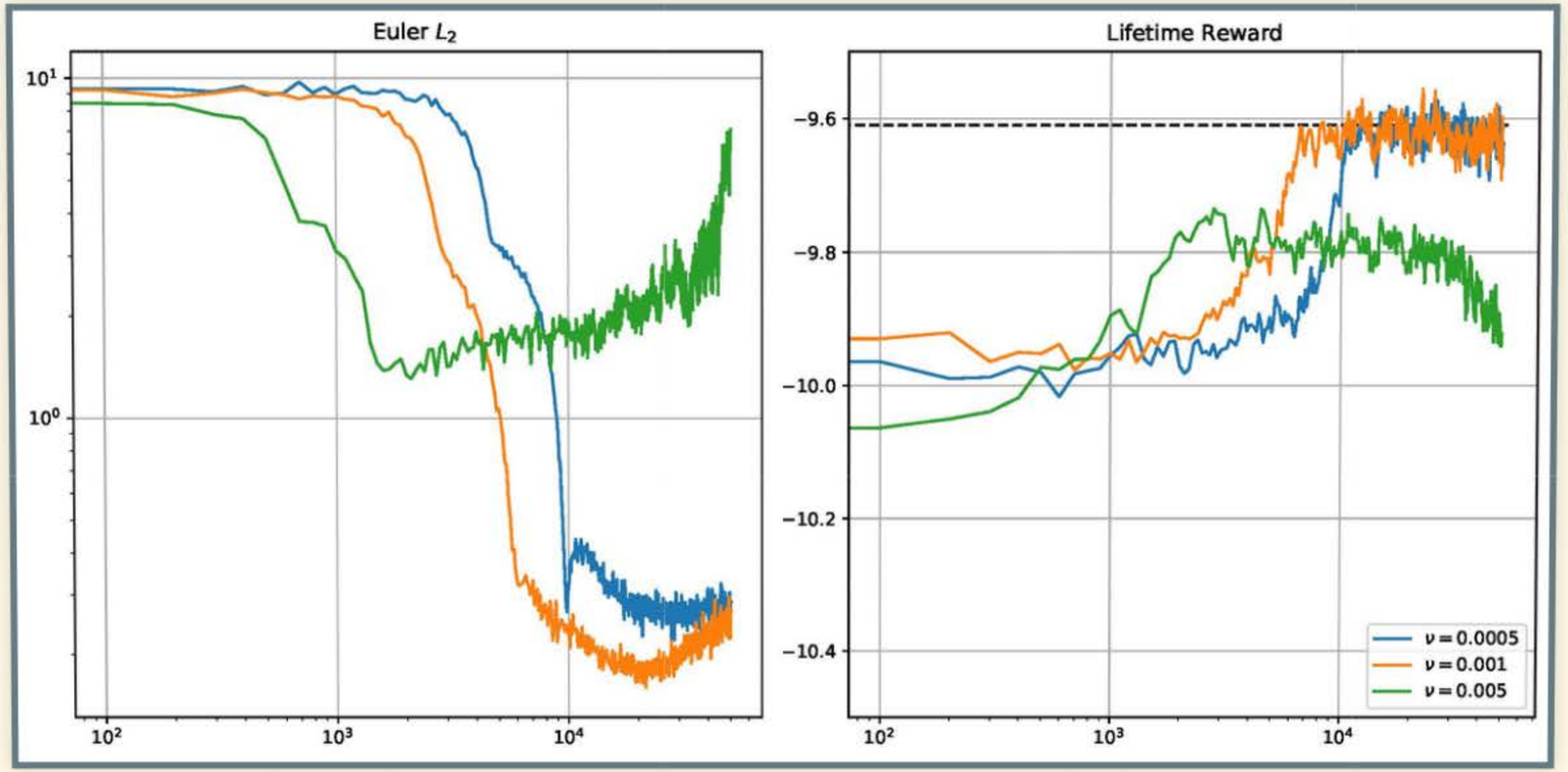
- Bellman equation contains the maximum operator to be constructed within the main iterative cycle
- We propose to deal with this operator by combining Bellman residuals minimization with the max operator in a single objective function:
- Select $V(\cdot;\theta_1)$ and $c(\cdot;\theta_2)$, draw (z,w) and define

$$\Xi(\theta) = E_{(z,w)}[V(z,w;\theta_1) - u(c) - \beta E_\varepsilon V(z',w';\theta_1)]^2 -$$

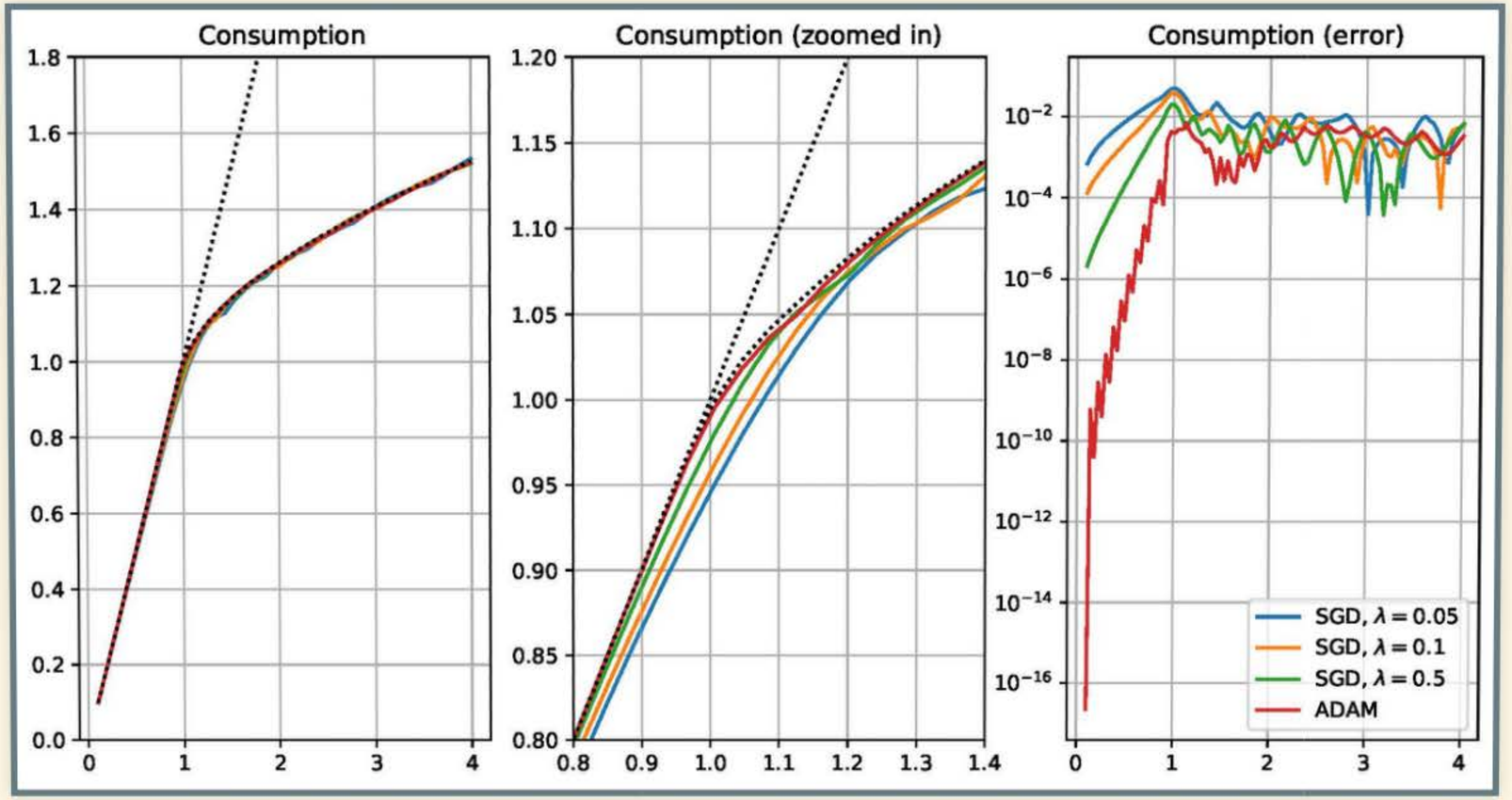
$$-\upsilon E_{(z,w)}[u(c) + \beta E_\varepsilon V(z',w';\theta_1)]$$

  - This is also not all-in-one-expectation operator!

OBJECTIVE 3: BELLMAN-RESIDUAL MINIMIZATION

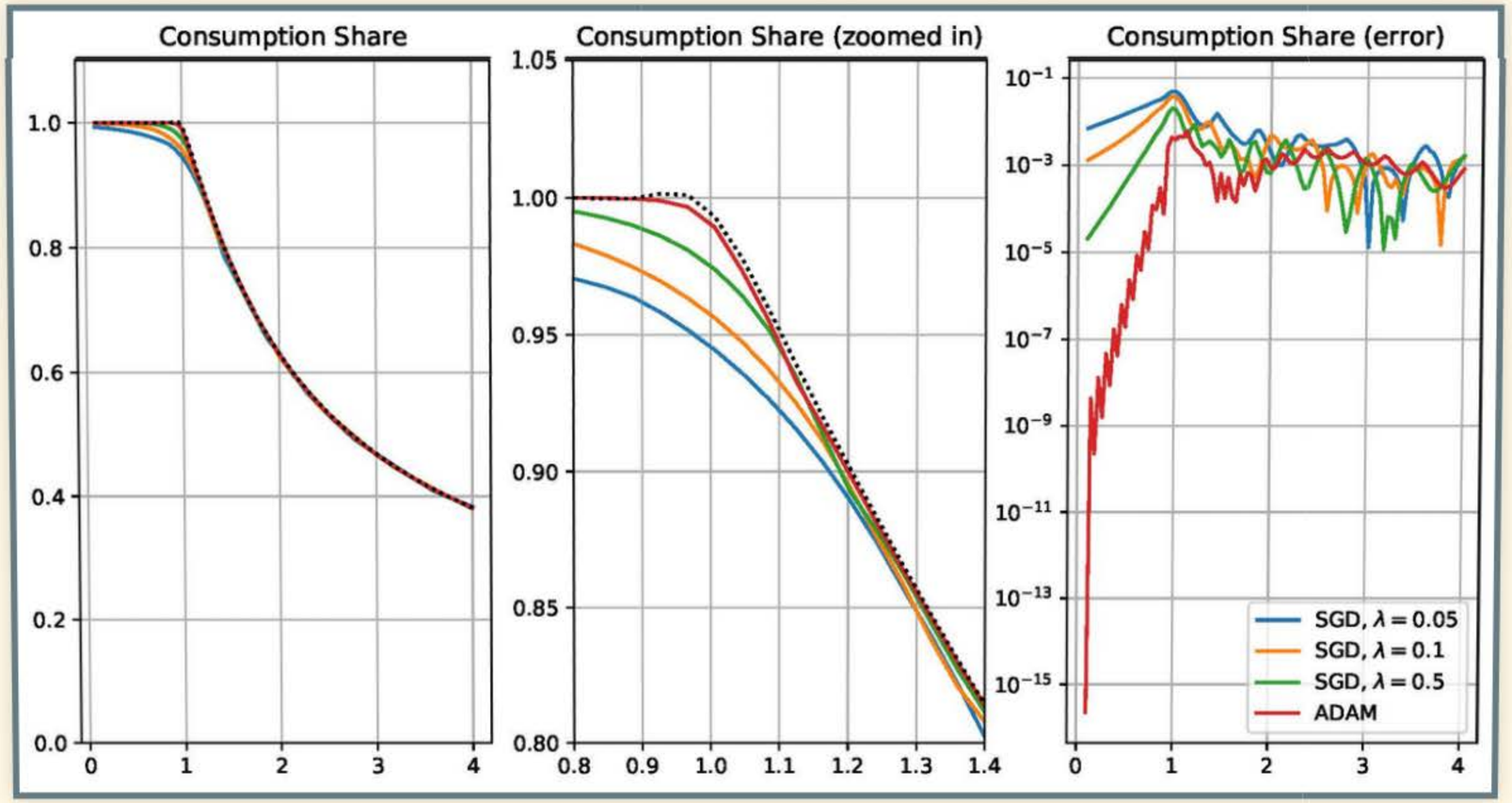


CONSUMPTION DECISION RULE



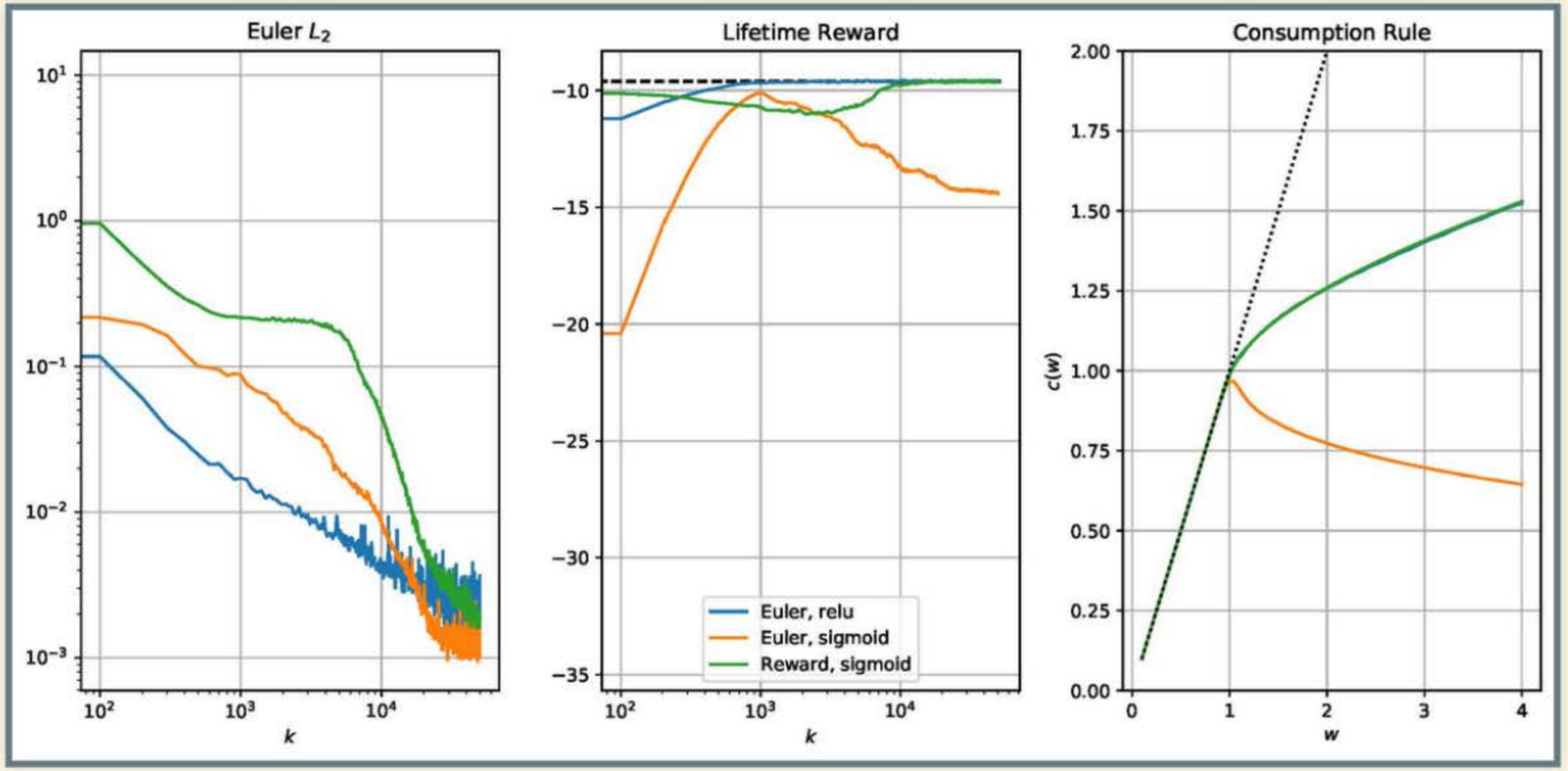
Training with the Euler-residual method

CONSUMPTION-SHARE DECISION RULE



Training with the Euler-residual method

ROLE OF ACTIVATION FUNCTIONS

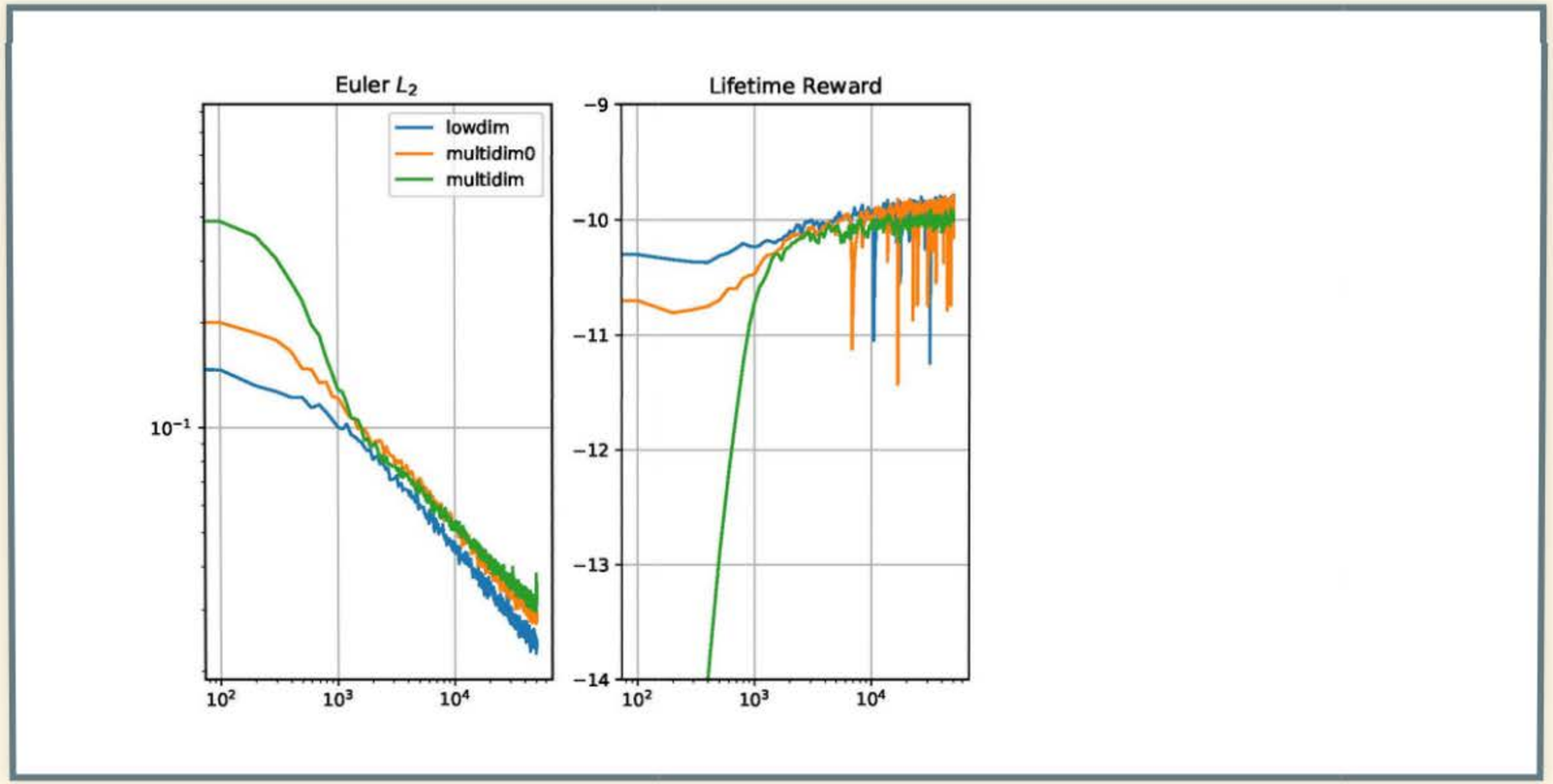


Using a sigmoid activation function (instead of relu)

DIMENSIONALITY REDUCTION AND ROBUSTNESS TO ILL CONDITIONING

- We compare 3 models:
  - Model 1: "multidim": 4 shocks $\{y_t, p_t, \varrho_t, \chi_t\}$
  - Model 2: "lowdim": 1 shock $\{y_t\}$
  - Model 3: "multidim0": 4 shocks $\{y_t, p_t, \varrho_t, \chi_t\}$ but set $\{p_t, \varrho_t, \chi_t\} = 0 \Rightarrow$ multicollinearity
- We find that our DL method:
  - **Scales great (linearly)!**
  - **Eliminates useless states (model reduction)!**
  - **Is not affected by ill conditioning!**

THE MULTISHOCK MODEL



Training with the reward-maximization method

ALL-IN-ONE EXPECTATION OPERATORS

- In lifetime-reward maximization, $E_{(z_0,w_0)}[E_{(\epsilon_1,\ldots,\epsilon_T)}u(\cdot)] = E_{(z_0,w_0,\epsilon_1,\ldots,\epsilon_T)}[u(\cdot)]$ because expectation operators are related linearly
- In Euler- and Bellman-residual minimization, $E_{(z,w)}(E_\epsilon[f_j(z,w,\epsilon)])^2 \neq E_{(z,w)}E_\epsilon[f_j(z,w,\epsilon)^2]$, i.e., expectation operators cannot be naturally combined.
- As a result, stochastic gradient is biased $E_{(z,w)}(E_\epsilon[\nabla f_j(z,w,\epsilon)])^2 \neq \nabla E_{(z,w)}(E_\epsilon[f_j(z,w,\epsilon)])^2$.

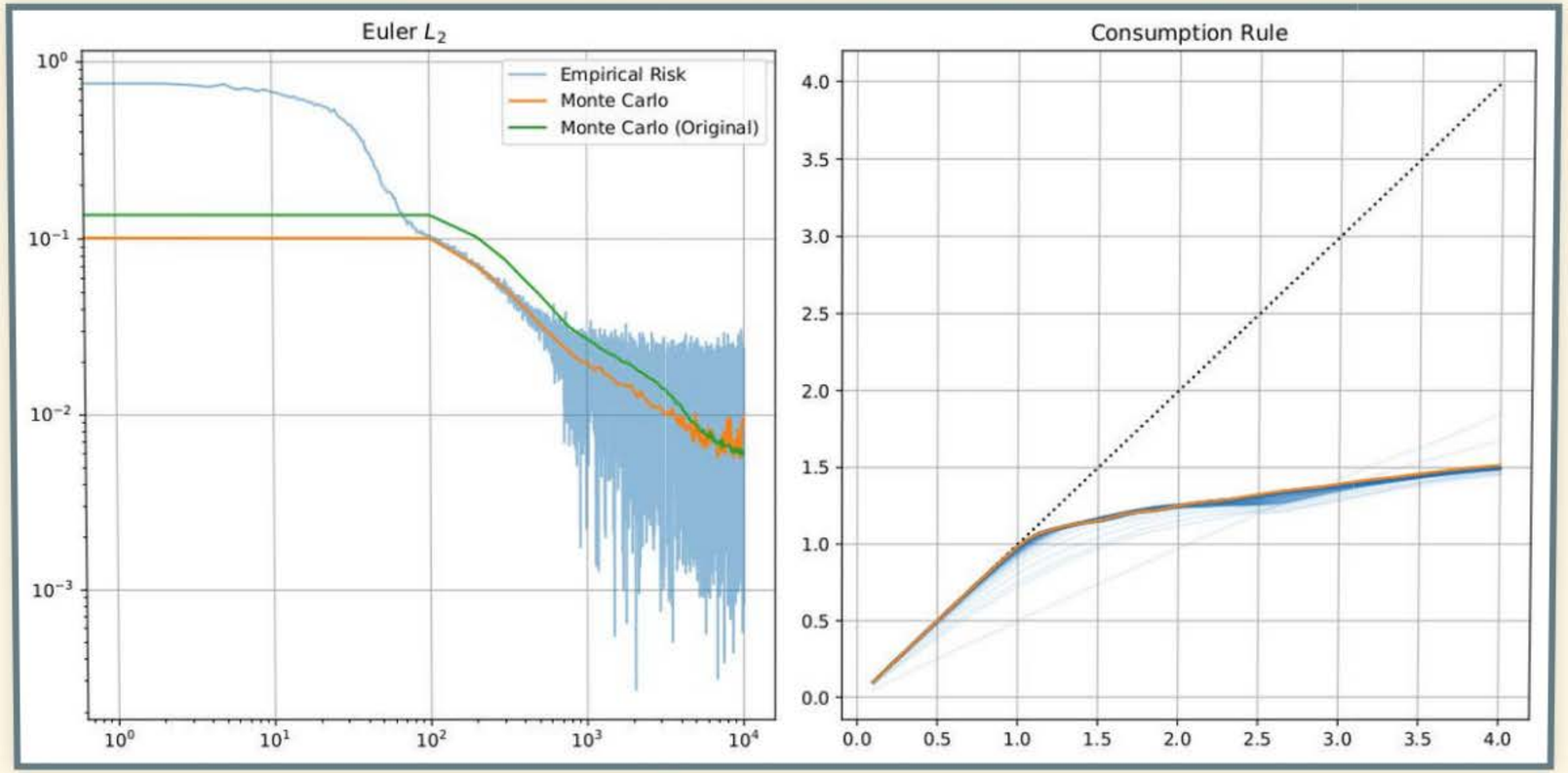
OUR CONTRIBUTION

- We came up with a simple technique that allows us to combine the expectation functions $E_{(z,w)}[\cdot]$ and $E_{\epsilon}[\cdot]$ in a single expectation operator in the presence of squares.
- Specifically, we use two independent random draws or two batches ϵ_1 and ϵ_2 for the two squared terms:
  $E_{\epsilon_1}[f(\epsilon_1)]E_{\epsilon_2}[f(\epsilon_2)] = E_{(\epsilon_1,\epsilon_2)}[f(\epsilon_1)f(\epsilon_2)]$
- With this method, the stochastic gradient descent method is unbiased.

ALL-IN-ONE EXPECTATION WITH FISHER-BURMEISTER

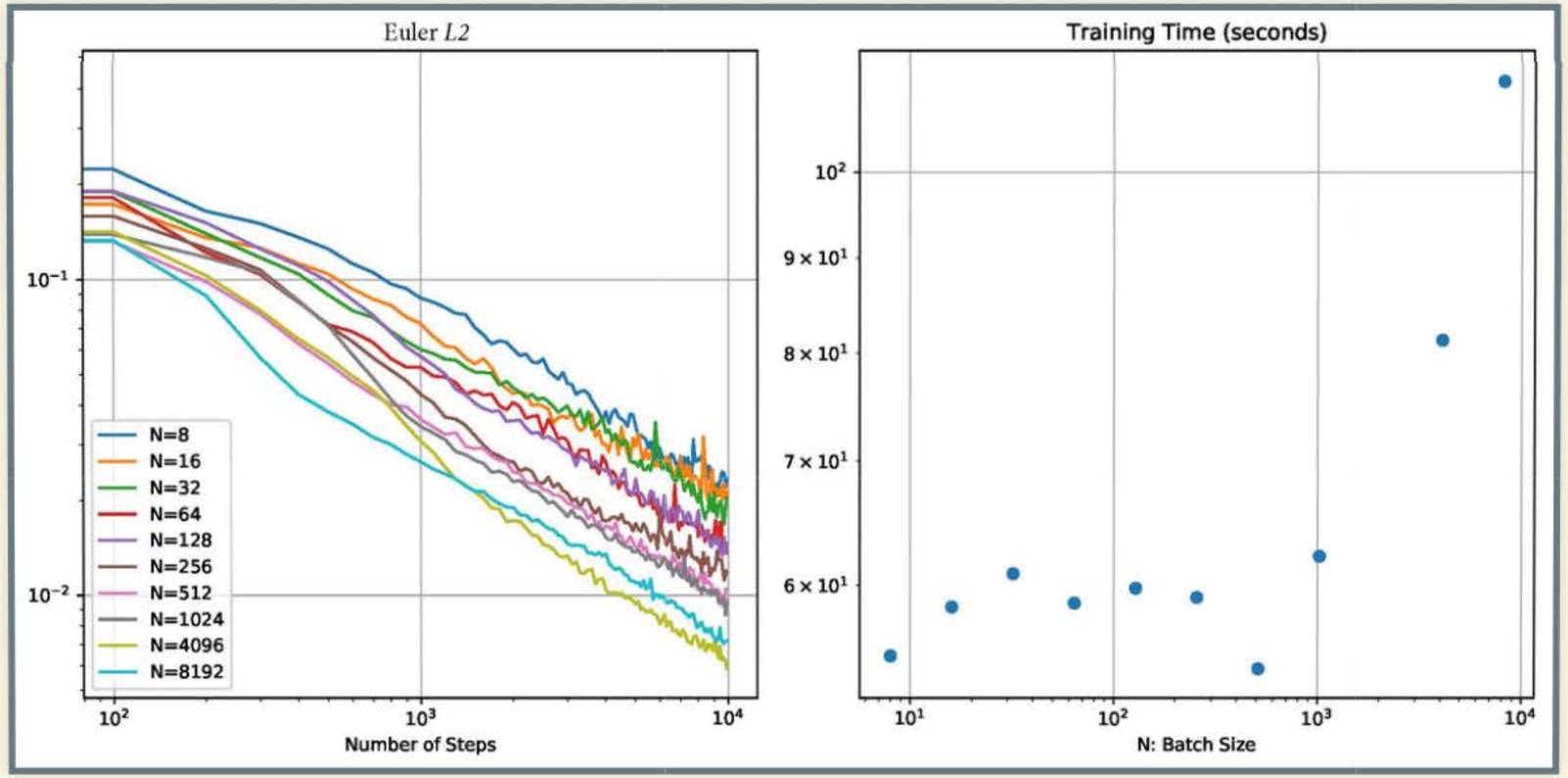
- First, introduce μ = supplementary variable representing the expectation function
- Then, define the Euler-residual operator
$E_{(z,w)}\{[\Psi(c-w,u'(c)-\mu)]^2+v[\beta re^{\varrho_t}E_\epsilon[u'(c'(\epsilon))]-\mu]^2\}$
where v=exogenous weight
- Finally, construct all-in-one-expectation operator
$\Xi(\theta)=E_{(z,w,\epsilon_1\epsilon_2)}\{[\Psi(c-w,u'(c)-\mu)]^2+$
$+v[\beta re^{\varrho_t}[u'(c'(\epsilon_1))]-\mu][\beta re^{\varrho_t}[u'(c'(\epsilon_2))]-\mu]\}$
- *Bellman-residual minimization* is similar

ALL-IN-ONE EXPECTATION OPERATOR (CONT.)



Training with Euler-residual method in the 4-shock model

SCALABILITY OF ALL-IN-ONE-EXPECTATION METHOD



Batch size varies from $N = 8$ to $N = 8192$ draws

KRUSELL AND SMITH (1998) MODEL

HETEROGENEOUS-AGENT SET UP

Agents $i = 1, \ldots, \ell$. Identical in fundamentals but differ in productivity and capital.

$$\max_{c_t^i, k_{t+1}^i} E_0 \left[\sum_{t=0}^{\infty} \beta^t u(c_t^i) \right]$$

$$\text{s.t. } c_t^i + k_{t+1}^i = R_t k_t^i + W_t z_t^i$$

$$z_{t+1}^i = \rho_z z_t^i + \sigma_z \epsilon_t^i, \text{ with } \epsilon_t^i \sim \mathcal{N}(0, 1)$$

$$k_{t+1}^i \geq 0, \text{ and } (k_0^i, z_0^i) \text{ given}$$

c_t^i, k_t^i are z_t^i = consumption, capital, labor productivity

HETEROGENEOUS-AGENT SET UP (CONT.)

- Cobb-Douglas production function $z_t k_t^{\alpha}$, with $k_t = \sum k_t^i$ = agg. capital, z_t = agg. productivity.

$$R_t = 1 - d + z_t \alpha k_t^{\alpha - 1} \ \& \ W_t = z_t (1 - \alpha) k_t^{\alpha - 1},$$

$$z_{t+1} = \rho z_t + \sigma \epsilon_t, \epsilon_t \sim \mathcal{N}(0, 1)$$

- State space: z_t & $\{k_t^i, z_t^i\}_{i=1}^{\ell}$.
- Krusell and Smith's (1998) method: replace distributions with a finite set of moments $m_t \Rightarrow$ approximate state space by $\{k_t^i, z_t^i, z_t, m_t\}$.

ALL-IN-ONE EXPECTATION

- We work directly with the actual state space.
- **Objective function:**

$$\Xi(\theta) = E_{(K,Z,z,\Sigma_1,\Sigma_2,\epsilon_{1,},\epsilon_2)}$$
$$\{[\Psi(c^i - w^i, u'(c^i) - \mu^i)]^2 + v[\beta[u'(c^{i'}(\Sigma_1, \epsilon_{1,}))] - \mu^i]$$
$$[\beta[u'(c^{i'}(\Sigma_2, \epsilon_{2,}))] - \mu^i]\},$$

$$K = (k_1^1, \ldots, k_1^\ell), Z = (z_1^1, \ldots, z_1^\ell), \Sigma_1 = (\epsilon_1^1, \ldots, \epsilon_1^\ell),$$
$$\Sigma_2 = (\epsilon_2^1, \ldots, \epsilon_2^\ell); \epsilon_{1,}, \epsilon_2 \text{ = agg. productivity shocks.}$$

DL SOLUTION ALGORITHM

Procedure

- i) draw initial state z_0 & $\{K_0, Z_0\} = \{k_0^i, z_0^i\}_{i=1}^{\ell}$;
- ii) compute aggregate capital k_0 and prices R_0, W_0;
- iii) train neural network for ℓ agents;
- iv) compute next period distribution

$$z_1 \text{ and } \{K_1, Z_1\} = \{k_1^i, z_1^i\}_{i=1}^{\ell}.$$

Proceed iteratively until convergence.

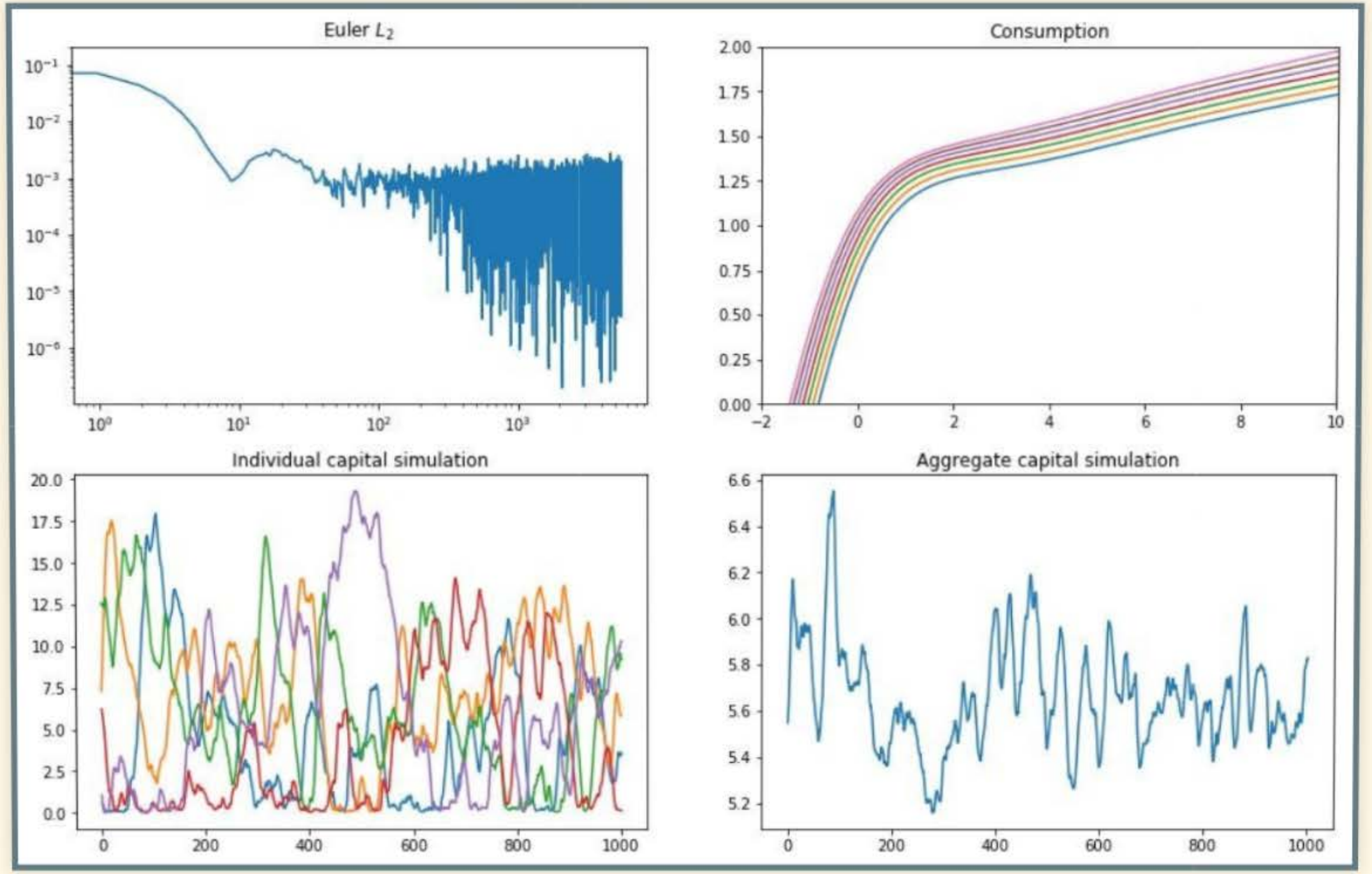
DL SOLUTION ALGORITHM (CONT.)

- **Ergodic-set domain**
  - deals with curse of dimensionality by simulating the state variables forward
- **Neural network handles multicollinearity**
  - include the state variables of agent i twice, $\varphi^i = \varphi(k_t^i, z_t^i, \{K_t, Z_t\}, z_t; \theta)$ but it is fine
- **Model reduction**
  - condenses $2\ell + 1 = 2,001$ state variables into 64 neurons in each hidden layer

CALIBRATION AND COMPUTATIONAL DETAILS

- $u(c) = \frac{c^{1-\gamma}-1}{1-\gamma}$ with $\gamma = 1$
- $\beta = 0.96; \rho = 0.9; \sigma = 0.07; \rho_z = 0.9$
  $\sigma_z = 0.4(1-\rho_z^2)^{1/2}$
- ADAM with batch size of 10 and learning rate of 0.001
- $K = 100,000$ iterations (equal to simulation length)

TRAINING WITH EULER-RESIDUAL METHOD, 5 AGENTS



SELECTED STATISTICS

ℓ	std(y)	corr(y,c)	Gini(k)	Bottom 40%	Top 20%	Top 1%	Time, sec.	R^2
1	1.51	0.858	-	-	-	-	235	0.999999
5	1.51	0.772	0.335	0.176	0.385	0.031	234	0.993473
10	1.51	0.595	0.391	0.144	0.428	0.036	254	0.995091
50	1.51	0.635	0.497	0.099	0.530	0.050	467	0.995284
100	1.51	0.658	0.450	0.121	0.486	0.047	1020	0.997600
500	1.51	0.462	0.484	0.096	0.506	0.052	7552	0.996554
1000	1.51	0.268	0.501	0.092	0.528	0.047	16435	0.995415

CONCLUSION

ENTHUSIASTIC VIEW

- Data scientists developed
  - powerful AI techniques for data analysis (DL)
  - general-purpose software for their efficient implementation (e.g., TensorFlow)
- Economists mostly develop own method & codes
  - some of which are surprisingly smart but typically, model-specific, not portable
- We show how to cast economic models into the state-of-the-art DL methods and software
- We hope this approach augments vastly the set of models tractable in economics

GRAIN OF SALT, BLACK MAGIC AND NO-FREE-LUNCH

However, we do not know whether our DL method is the best possible for all economic models.

- **First**, *neural network* is a promising approximator but has a large number of parameters and is highly non-linear.
- **Second**, *Monte Carlo integration* has a low, square-root rate of convergence.
- **Third**, *stochastic optimization* is magical but its convergence rate is lower and not guaranteed.
- **Finally**, there are other promising AI technologies that can be used for solving economic models.

DO NOT RETIRE COMPUTATIONAL ECONOMISTS YET!

- The idea that there is an all-purpose procedure, which will remove the curse of dimensionality, without any trade-offs, for any kind of models, and will produce highly precise and uniformly accurate solution *belongs to magical thinking*
- The human input is still critical to see trade-offs.
- The bottom line is that it is too early to retire all computational economists at the moment.
- At least, not until they figure out what type of AI works best for economic models.

Sync/BondsAll | !SR_Bonds - Ju | !SR_Bonds - Sh | !Training - Jupy | Comparison-B | Bonds_Hank - | Google Drive | model.py - Jup | Show Results - | Bolsa Madrid | Will artificial in

file:///D:/Google Drive New/Deep_Learning_SL/ASSA2021/PresentationASSA2021.html#/11

THANK YOU!

ESP 6:19 PM 12/23/2020