Deep Learning Classification: Modeling Discrete Labor Choice

Lilia Maliar and Serguei Maliar

January, 2022

ASSA 2022

Discrete- versus continuous-set choices

- Macroeconomic models are generally built on continuous-set choices.
- For example, the agent can distribute wealth in any proportion between consumption and savings or she can distribute time endowment in any proportion between work and leisure.
- But certain economic choices are discrete: the agent can either buy a house or not, be either employed or not, either retire or not, etc.

The progress in modeling discrete choices is still limited!

The results in the present paper

- We introduce a deep learning classification (DLC) method that solves models with both continuous-set and discrete-set choices.
- To solve for continuous-set choices:
  - we parameterize decision functions with a deep neural network;
  - and we find the coefficients of the neural network (biases and weights) to satisfy the model's equations.
- Our main novelty is a classification method for constructing discrete-set choices.
- We define a state-contingent probability function that:
  - for each feasible discrete choice, gives the probability that this specific choice is optimal;
  - we parameterize the probability function with a deep neural network;
  - and we find the network parameters to satisfy the optimality conditions for the discrete choices.

An illustration from data science: image recognition

- Consider the image recognition problem–a typical classification problem in data science.
- For example, a machine classifies images into cats, dogs and sheep.
- We parameterize the probabilities of the three classes with a deep neural network.
- The machine is given a collection of images and is trained to minimize the cross-entropy loss (which is equivalent to maximizing the likelihood function) that ensures the correct classification of images; see Goodfellow, Bengio and Courville (2016) for a survey of classification methods in data science.

Classification method for discrete choices in economics

- Our classification method in macroeconomics is analogous to the above image-recognition analysis.
- For example, we use a deep neural network to parameterize the probabilities of being full-time employed, part-time employed and unemployed.
- The machine is given a collection of employment choices conditional on state and is trained to maximize the likelihood function that those choices are optimal.
- The same idea can be applied for analyzing the models with retirement, default, house purchase, etc.

Remark:

- The earlier literature on indivisible labor (e.g., Rogerson (1996) and Hansen (1994)) construct discrete choice by introducing lotteries.
- Our probabilities have totally different meaning: they indicate which discrete choices is most likely to be optimal and hence, is selected.

Problems with high dimensionality

- The DLC classification solution method we propose can be used to solve small-scale representative agent models.
- However, the power of deep learning consists in its ability to solve large-scale applications that are intractable with conventional solution methods.
- To illustrate these remarkable capacities of the DLC method, we solve Krusell and Smith's (1998) model in which the agents face indivisible labor choices.

The literature on heterogeneous agent models

- Krusell and Smith's (1998) model is computationally challenging even in the absence of discrete choices.
- The state space may include thousands of state variables of heterogenous agents and is prohibitively large.
- To make the model tractable, Krusell and Smith (1998) replace distributions with few aggregate moments but that approach does not always work.
- Several recent papers use linearization and perturbation to simplify the analysis of equilibrium in heterogeneous-agent models, including Reiter (2010), McKay and Reis (2016), Childers (2016), Boppart et al. (2018), Mertens and Judd (2017), Ahn et al (2018), Winberry (2018), Bayer and Luetticke (2020)
- Reiter (2019) provides for a thoughtful discussion of that literature.

DLC method

- A distinctive feature of our DLC method is that it does not rely on moments, linearization, perturbation or any other pre-designed reduction of the state space.
- It works with the actual state space consisting of all individual and aggregate state variables – we let deep neural network to choose how to condense large sets of state variables into much smaller sets of features.
- Our code is written using Google's TensorFlow platform – deep learning software that led to many ground breaking applications in data science – and is it tractable in models with thousands of state variables.

Relation to the literature on deep learning in economics

- Our DLC method is related to recent papers on deep learning, including Duarte (2018), Villa and Valaitis (2019), Fernández-Villaverde, Hurtado, and Nuño (2019), Azinović, Luca and Scheidegger (2019), Lepetyuk, Maliar and Maliar (2020) and especially, Maliar, Maliar and Winant (2018, 2019, 2021).
- However, this literature does not analyze models with discrete choices, which is the main subject of the present paper.

Relation to the literature on discrete choices

- There are numerous methods in econometrics for estimating discrete-choice models but these methods are limited to statistic applications; see Train (2009) for a review.
- The macro literature with discrete choices includes Chang and Kim (2007) and Chang, Kim, Kwon and Rogerson (2019) who solve a similar model by using Krusell and Smith (1998) analysis.
- Iskhakov, Jørgensen, Rust and Schjerning (2017) developed an endogenous grid method with taste shocks that is designed to deal with discrete choices in dynamic environment.
- In the context of Carroll's (2005) analysis, that paper suggests to apply logistic smoothing to the kinks by transferring the problem into the choice probability space via the taste shocks.
- In contrast, we do not attempt to smooth the kinks but instead to accurately approximate such kinks by using the-state-of-the-art deep learning classification method.

Applications: Krusell and Smith's (1998) model

- a version of Krusell and Smith's (1998) model with continuous choices (i.e., divisible labor);
- an indivisible-labor version with 2 discrete labor states (employed and unemployed);
- an indivisible-labor version with 3 discrete labor states (employed, unemployed and part-time employed agent).

The model

- Heterogeneous agents $i = 1, ..., \ell$. Each agent i solves

$$\max_{\left\{c_t^i, k_{t+1}^i, n_t^i\right\}_{t=0}^{\infty}} E_0 \left[\sum_{t=0}^{\infty} \beta^t u\left(c_t^i, n_t^i\right)\right]$$
$$\text{s.t. } c_t^i + k_{t+1}^i = R_t k_t^i + W_t v_t^i n_t^i,$$
$$n_t \in N,$$
$$\ln v_{t+1}^i = \rho_v \ln v_t^i + \sigma_v \epsilon_t^i \text{ with } \epsilon_t^i \sim \mathcal{N}(0,1),$$
$$k_{t+1}^i \geq \overline{k},$$

  where c_t^i, n_t^i, k_t^i and v_t^i are consumption, hours worked, capital and idiosynratic labor productivity; $\beta \in (0,1)$ is the discount factor; $\rho_v \in (-1,1)$ and $\sigma_v \geq 0$; and initial condition $\left(k_0^i, v_0^i\right)$ is given. The capital choice is restricted by a borrowing limit $\overline{k} \leq 0$.

- The three different versions of the model are distinguished by the set of allowable labor choices N.

Production side

- The production side of the economy is described by a Cobb-Douglas production function $\exp(z_t)\, k_t^{\alpha-1} h_t^{1-\alpha}$, where $k_t = \sum_{i=1}^{\ell} k_t^i$ is aggregate capital, $h_t = \sum_{i=1}^{\ell} v_t^i n_t^i$ is aggregate efficiency labor, and z_t is an aggregate productivity shock following a first-order autoregressive process,

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

  where $\rho_z \in (-1,1)$ and $\sigma_z \geq 0$.
- The interest rate R_t and wage W_t are given by

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

  where $d \in (0,1]$ is the depreciation rate.

Kuhn-Tucker condition

- The Kuhn-Tucker condition with respect to capital is

$$\mu_t^i \delta_t^i = 0,$$

  where $\delta_t^i \equiv k_{t+1}^i - \overline{k} \geq 0$ is the distance to the borrowing limit, and $\mu_t^i \geq 0$ is the Lagrange multiplier

$$\mu_t^i \equiv u_1 \left(c_t^i, n_t^i \right) - \beta E_t \left[u_1 \left(c_{t+1}^i, n_{t+1}^i \right) R_{t+1} \right],$$

  where u_1 denotes a first-order partial derivative of function u with respect to the first argument.
- Whenever $\delta_t^i > 0$, the agent is not at the borrowing limit, i.e., $k_{t+1}^i > \overline{k}$, so the Euler equation must hold with equality leading to $\mu_t^i = 0$, and whenever the Euler equation does not hold with equality, it must be that the agent is at the borrowing constraint $\delta_t^i = 0$

Three different version of the model

We consider three versions of the model that differ in the set of allowable labor choices $n_t \in N$:

i)	divisible labor model	$N = [0, L]\,,$
ii)	indivisible labor model	$N = \{0, \overline{n}\}\,,$
iii)	three-state employment model	$N = \{0, \underline{n}, \overline{n}\}\,,$

Divisible labor model

- To characterize labor choice, we assume that the utility function takes the form

$$u\left(c,n\right)=\frac{c^{1-\gamma}-1}{1-\gamma}+B\frac{\left(L-n\right)^{1-\eta}-1}{1-\eta},$$

  where γ, $\eta>0$ and L is the total time endowment.

- We normalize time to L instead of the conventional normalization to 1 because it helps to calibrate the divisible and indivisible labor models to the same steady state.
- The labor choice is characterized by a FOC

$$n_t^i=L-\left[\frac{c_i^{-\gamma}W_t v_t^i}{B}\right]^{-1/\eta}.$$

Indivisible labor model with 2 states

The agent chooses to be employed ($n_t^i = \overline{n}$) or unemployed ($n_t^i = 0$) depending on which of the two choices leads to a higher continuation value, i.e.,

$$\begin{aligned} n_t^i &= \overline{n} \text{ if } V^E = \max\left\{V^E, V^U\right\} \\ n_t^i &= 0 \text{ otherwise.} \end{aligned}$$

where V^E and V^U denote value functions of the agent in the employed and unemployed states, respectively.

Indivisible labor model with 3 states

The three employment states, $n_t^i = \overline{n}$, $n_t^i = \underline{n}$ and $n_t^i = 0$, correspond to full-time unemployment, part-time employment and unemployment, respectively,

$$\begin{aligned} n_t^i &= \overline{n} \text{ if } V^{FT} = \max\left\{V^U,\ V^{FT}, V^{PT}\right\} \\ n_t^i &= \underline{n} \text{ if } V^{PT} = \max\left\{V^U,\ V^{FT}, V^{PT}\right\} \\ n_t^i &= 0 \text{ otherwise} \end{aligned}$$

where V^{FT}, V^{PT} and V^U denote value functions of full-time employed, part-time employed and unemployed agents, respectively.

Deep learning method for divisible labor model

Deep learning method for divisible labor model

The state space of Krusell and Smith's (1998) model has $2\ell + 1$ state variables; for example, with $\ell = 1,000$, the state space has $2,001$ state variables. To deal with so large dimensionality, we rely on a combination of techniques introduced in Maliar et al. (2018, 2019, 2021), including:

1. stochastic simulation that allows us to restrict attention to the ergodic set in which the solution "lives";
2. multilayer neural networks that perform model reduction and help deal with multicollinearity;
3. a (batch) stochastic gradient descent method that reduces the number of function evaluations by operating on random grids;
4. a Fischer-Burmeister function that effectively approximates the kink;
5. most importantly, "all-in-one expectation operator" that allows us to approximate high-dimensional integrals with just 2 random draws (or batches) on each iteration.
6. TensorFlow – a Google data science platform that is used to facilitate the remarkable data-science applications such as image and speech recognition, self driving cars, etc.

Stochastic simulation - ergodic set domain

- Under normally distributed shocks, stochastic simulation typically have a shape of a hypersphere (hyperoval)

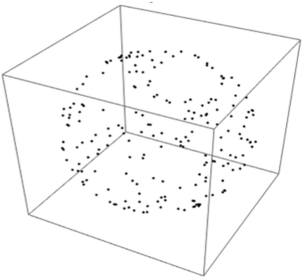

Figure 1. Hypercube versus hypersphere.

- The ratio of a volume of a hypersphere to that of an enclosing hypercube is an infinitesimally small number in high-dimensional applications; for example, for a 30-dimensional case, it is 10^{-14}; see Judd, Maliar and Maliar (2011) for a discussion.

Neural networks

We use neural networks for parameterizing decision and value functions instead of more conventional approximation families like polynomial functions:

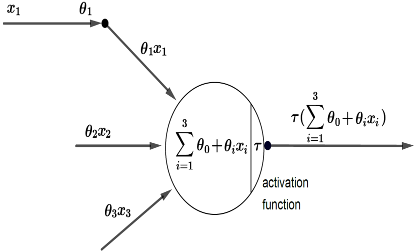

Figure 2a. Artificial neuron.

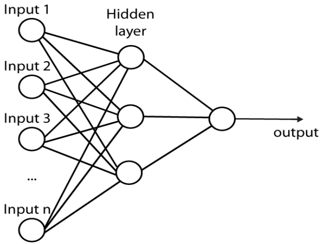

Figure 2b. Neural network.

In Figure 1a, the circle represents an artificial neuron that receives 3 signals (inputs) x_1, x_2 and x_3. In Figure 1b, we combine multiple neurons into a neural network.

Activation functions

The activation function that we use in our benchmark experiments is a sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\theta_0+\theta_1 x_1+\theta_2 x_2+\ldots+\theta_n x_n}}$.

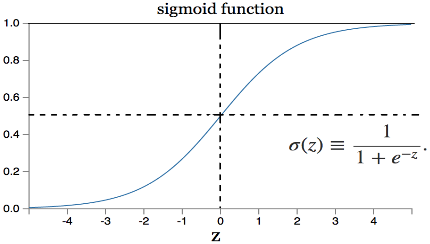

Figure 3. Sigmoid function.

The sigmoid function has two properties: First, its derivative can be inferred from the function itself $\sigma'(x) = \sigma(x)(1-\sigma(x))$. Second, it maps a real line into a unit interval $\sigma : \mathbb{R}^n \rightarrow [0, 1]$ which makes it bounded between 0 and 1.

Parameterization of decision functions

- We solve for two decision functions–hours worked $\frac{n_t^i}{L}$ and the fraction of wealth that goes to consumption $\frac{c_t^i}{w_t^i}$ which we parameterized by a sigmoid function
$$\sigma\left(\zeta_0+\varphi\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t; \theta\right)\right),$$
where $\varphi(\cdot)$ is a multilayer neural network parameterized by a vector of coefficients θ (weights and biases), $\sigma(z)=\frac{1}{1+e^{-z}}$ is a sigmoid function and ζ_0 is a constant term.
- In addition, we parameterize the Lagrange multiplier μ_t^i associated with the borrowing constraint using an exponential activation function
$$\exp\left(\zeta_0+\varphi\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t; \theta\right)\right).$$
The exponential activation function ensures that the Lagrange multiplier is always non-negative.
- Since the agents are identical in fundamentals, the above three $2\ell+1$–dimensional decision functions are sufficient to characterize the choices of all ℓ heterogeneous agents.

Model reduction

- Our DLC solution method aims at solving models with thousands of state variables by using model reduction.
- It condenses the information from a large number of inputs into a smaller number of neurons in the hidden layers, making it progressively more abstract and compact.
- This procedure is similar to a photo compression or principal component transformation when a large dataset is condensed into a smaller set of principal components without losing essential information; see Judd, Maliar and Maliar (2011) for a discussion of model reduction using principal-component analysis.
- Krusell and Smith (1998) proposed one specific model reduction method, namely, they approximate the distribution with just one moment – the mean.
- If Krusell and Smith's (1998) analysis is the most efficient representation of the state space, the neural network will also find it.
- However, the neural network will consider many other possible ways of extracting the information from the distributions and condensing it in a relatively small set of hidden layers trying to find the best one.

Objective function for deep learning

- The objective is to minimize the squared residuals in three model's conditions:

$$\Xi(\theta) \equiv E_{(K_t,Y_t,z_t)} \left\{ \left[\Psi^{FB} \left(1 - \frac{c_t^i}{w_t^i}, 1 - \mu_t^i \right) \right]^2 \right.$$

$$+ \varpi_n \left[n_t^i - \left(L - \left[\frac{\left(c_t^i\right)^{-\gamma} W_t v_t^i}{B} \right]^{-1/\eta} \right) \right]^2$$

$$\left. + \varpi_\mu \left[\frac{\beta E_{(\Sigma_{t+1},\epsilon_{t+1})} \left[\left(c_{t+1}^i\right)^{-\gamma} R_{t+1} \middle| \Sigma_{t+1}, \epsilon_{t+1} \right]}{\left(c_t^i\right)^{-\gamma}} - \mu_t^i \right]^2 \right\},$$

where $K \equiv \left(k^1, ..., k^\ell\right)$ and $Y \equiv \left(v^1, ..., v^\ell\right)$ are state variables; z_t is aggregate productivity; $\Sigma_{t+1} \equiv \left(\epsilon_{t+1}^1, ..., \epsilon_{t+1}^\ell\right)$ the individual productivity shocks; ϵ_{t+1} is the aggregate productivity shock; and

$$\Psi^{FB}(a, b) = a + b - \sqrt{a^2 + b^2},$$

is a $\Psi^{FB}(a, b) = 0$ is a Fisher-Burmeister objective function is equivalent to Kuhn Tucker conditions.

All in one expectation operator

- The constructed objective function $\Xi(\theta)$ is not convenient because it contains a square of expectation $\left[E_{(\Sigma_{t+1},\epsilon_{t+1})}\left[\cdot\right]\right]^2$ nested inside another expectation $E_{(K_t,Y_t,z_t)}\left[\cdot\right]$.
- Constructing two nested expectation operators is costly because the inner expectation operator $E_{(\Sigma_{t+1},\epsilon_{t+1})}\left[\cdot\right]$ has high dimensionality; if $\ell = 1,000$, it is $1,001$-dimensional integral.
- This task would be simplified enormously if we could combine the two expectation operators but it is not possible $E_{(K_t,Y_t,z_t)}\left[\left[E_{(\Sigma_{t+1},\epsilon_{t+1})}\left[\cdot\right]\right]^2\right] \neq E_{(K_t,Y_t,z_t)}E_{(\Sigma_{t+1},\epsilon_{t+1})}\left[\left[\cdot\right]^2\right]$.
- Maliar et al. (2021) propose a simple but powerful technique, called *all-in-one* (AiO) expectation operator, that can merge the two expectation operators into one.
- They replace the squared expectation function $\left[E_{(\Sigma_{t+1},\epsilon_{t+1})}\left[\cdot\right]\right]^2$ under one random draw $(\Sigma_{t+1},\epsilon_{t+1})$ with a product of two expectation functions $\left[E_{(\Sigma'_{t+1},\epsilon'_{t+1})}\left[\cdot\right]\right] \times \left[E_{(\Sigma''_{t+1},\epsilon''_{t+1})}\left[\cdot\right]\right]$ under two uncorrelated random draws $\left(\Sigma'_{t+1},\epsilon'_{t+1}\right)$ and $\left(\Sigma''_{t+1},\epsilon''_{t+1}\right)$.
- Since the two random draws are uncorrelated, the expectation operator can be taken outside of the expectation function.

The objective function under AiO expectation operator

$$
\begin{aligned}
\Xi(\theta) \equiv E_{\left(K_t, Y_t, z_t, \Sigma'_{t+1}, \epsilon'_{t+1}, \Sigma''_{t+1}, \epsilon''_{t+1}\right)} \Bigg\{ \left[\Psi^{FB}\left(1 - \frac{c_t^i}{w_t^i}, 1 - \mu_t^i\right) \right]^2 \\
+ \varpi_n \left[n_t^i - \left(L - \left[\frac{\left(c_t^i\right)^{-\gamma} W_t v_t^i}{B} \right]^{-1/\eta} \right) \right]^2 + \varpi_\mu \times \\
+ \left[\frac{\beta \left[\left(c_{t+1}^i\right)^{-\gamma} R_{t+1} \middle| \Sigma'_{t+1}, \epsilon'_{t+1} \right]}{\left(c_t^i\right)^{-\gamma}} - \mu_t^i \right] \left[\frac{\beta \left[\left(c_{t+1}^i\right)^{-\gamma} R_{t+1} \middle| \Sigma''_{t+1}, \epsilon''_{t+1} \right]}{\left(c_t^i\right)^{-\gamma}} - \mu_t^i \right]
\end{aligned}
$$

Thus, we are able to represent the studied model as an expectation function across a vector of random variables $\left(K_t, Y_t, z_t, \Sigma'_{t+1}, \epsilon'_{t+1}, \Sigma''_{t+1}, \epsilon''_{t+1}\right)$; see Maliar et al. (2021) for a discussion and further applications of the AiO expectation operator.

Training: gradient descent, batches and parallel computing

- Given that AiO is an expectation function, we can bring the gradient operator inside by writing $\nabla_\theta \Xi(\theta) = \nabla_\theta E\left[\xi\left(\omega;\theta\right)\right] = E\left[\nabla_\theta \xi\left(\omega;\theta\right)\right]$, where ∇_θ is a gradient operator.
- The latter expectation function can be approximated by a simple average across Monte Carlo random draws $E\left[\nabla_\theta \xi\left(\omega;\theta\right)\right] \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta \xi\left(\omega_n;\theta\right)$, where ω_n denotes a specific realization of the vector of random variables.
- Thus, the gradient descent method can be implemented as

$$\theta \leftarrow \theta - \lambda \nabla_\theta \Xi(\theta) \qquad \text{with} \qquad \nabla_\theta \Xi(\theta) \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta \xi\left(\omega_n;\theta\right),$$

  where θ and λ are the parameter vector and learning rate, respectively.
- Thus, we implement a cheap computation of the gradient of the integrand instead of computing far more expensive gradient of the expectation function. TensorFlow and PyTorch can compute such a gradient using a symbolic differentiation, which facilitates an the implementation of parallel computation.

Dealing with multicollinearity

- In the arguments of approximating functions, the state variables of agent i appear twice $\varphi\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t; \theta\right)$ because they enter both as variables of agent i and as an element of the distribution.
- This repetition implies perfect collinearity in explanatory variables, so that the inverse problem is not well defined.
- Such a multicollinearity would break down a conventional least-squares method which solves the inverse problem (since an inverse of a matrix with linearly dependent rows or columns does not exist).
- However, neural networks are trained by using the gradient-descent method that avoids solving an inverse problem. As a result, neural networks can learn to ignore redundant colinear variables; see Maliar et al. (2021) for numerical illustrations and a discussion.

Algorithm 1: Deep learning for divisible labor model

Algorithm 1: Deep learning for divisible labor model.
Step 0: (Initialization).
Construct initial state of the economy $\left(\left\{k_0^i, v_0^i\right\}_{i=1}^{\ell}, z_0\right)$ and parameterize three decision functions by a neural network with three outputs $$\left\{\frac{n_t^i}{L}, \frac{c_t^i}{w_t^i}\right\} = \sigma\left(\zeta_0 + \varphi\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t; \theta\right)\right),$$ $$\mu_t^i = \exp\left(\zeta_0 + \varphi\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t; \theta\right)\right),$$ where $w_t^i \equiv R_t k_t^i + W_t v_t^i n_t^i$ is wealth; μ_t^i is Lagrange multiplier associated with the borrowing constraint; $\varphi(\cdot)$ is a neural network; $\sigma(z) = \frac{1}{1+e^{-z}}$ is a sigmoid (logistic) function; ζ_0 is a constant; θ is a vector of coefficients.

Algorithm 1: Deep learning for divisible labor model (cont)

Algorithm 1: Deep learning for divisible labor model.
Step 1: (Evaluation of decision functions).
Given state $\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t\right) \equiv s_t^i$, compute n_t^i, μ_t^i, $\frac{c_t^i}{w_t^i}$ from the neural networks, find the prices R_t and W_t; and find k_{t+1}^i from the budget
constraint for all agents $i = 1, ..., \ell$.
Step 2: (Construction of Euler residuals).
Draw two random sets of individual productivity shocks $\Sigma_1 = \left(\epsilon_1^1, ..., \epsilon_1^{\ell}\right)$, $\Sigma_2 = \left(\epsilon_2^1, ..., \epsilon_2^{\ell}\right)$ and two aggregate shocks $\epsilon_{1,}$, ϵ_2, and construct Euler residuals $\Xi(\theta) = \left\{\left[\Psi^{FB}\left(1 - \frac{c_t^i}{w_t^i}, 1 - \mu_t^i\right)\right]^2 + \varpi_n \left[n_t^i - \left(L - \left[\frac{\left(c_t^i\right)^{-\gamma} W_t v_t^i}{B}\right]^{-1/\eta}\right)\right]^2 + \varpi_\mu \left[\frac{\beta\left[\left(c_{t+1}^i\right)^{-\gamma} R_{t+1} \middle\| \Sigma'_{t+1}, \epsilon'_{t+1}\right]}{\left(c_t^i\right)^{-\gamma}} - \mu_t^i\right]\left[\frac{\beta\left[\left(c_{t+1}^i\right)^{-\gamma} R_{t+1} \middle\| \Sigma''_{t+1}, \epsilon''_{t+1}\right]}{\left(c_t^i\right)^{-\gamma}} - \mu_t^i\right]\right\}$, where ϖ_n, ϖ_μ are given weights and $\Psi^{FB}(a,b) = a + b - \sqrt{a^2+b^2}$ is a Fischer-Burmeister function.

Algorithm: Deep learning for divisible labor model (cont.)

Algorithm 1: Deep learning for divisible labor model.
Step 3: (Training).
Train the neural network coefficients θ to minimize the residual function $\Xi(\theta)$ by using a stochastic gradient descent method $\theta \leftarrow \theta - \lambda \nabla_\theta \Xi(\theta)$ with $\nabla_\theta \Xi(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta \xi\left(\omega_n; \theta\right)$, where $n = 1, ..., N$ denotes batches.
Step 4: (Simulation).
Move to $t+1$ by using endogenous and exogenous variables of Step 3 under $\Sigma_1 = \left(\epsilon_1^1, ..., \epsilon_1^\ell\right)$ and ϵ_1 as a next-period state $\left(\left\{k_{t+1}^i, v_t^i\right\}_{i=1}^{\ell}, z_{t+1}\right)$.

Calibration

- For our numerical analysis, we assume $\alpha = 0.36$; $d = 0.08$; $\beta = 0.96$; $\rho = 0.9$; $\sigma = 0.1$; $\rho_z = 0.9$; $\sigma_z = 0.21$; and $\overline{k} = 0$ – these values are in line with the literature, e.g., Chang and Kim (2007), Reiter (2010, 2019), Chang et al. (2019).
- We perform training using the *ADAM* stochastic gradient descent method with the batch size of 100 and the learning rate of 0.001.
- We fix the number of iterations (which is also a simulation length) to be $K = 100,000$.
- The choice of these parameters must ensure both convergence and low running time and it reflects our experience in constructing deep learning approximations.
- Finally, we study numerically the role of the elasticities γ and η of the utility function by performing a sensitivity analysis..

Training errors and running time

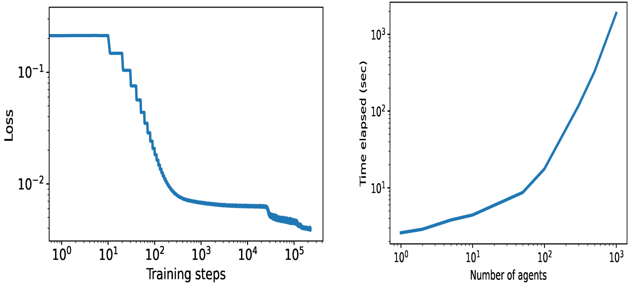

Figure 4. Training errors and running time for divisible labor model.

The solution for divisible labor model

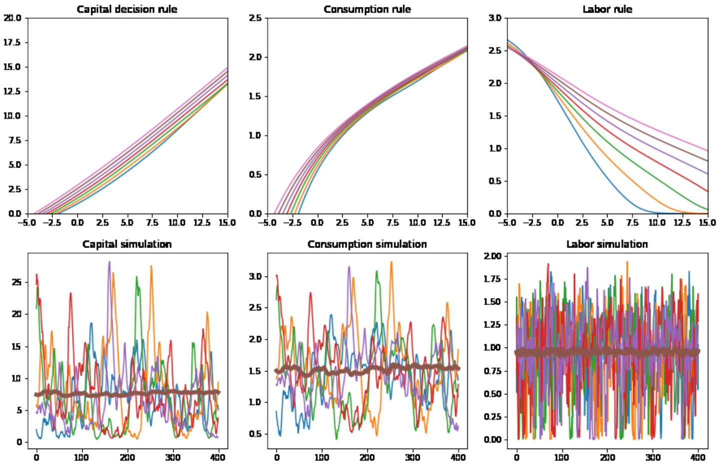

Figure 5. Solution to divisible labor mode.

Deep learning method for indivisible labor model

Logistic regression

Let us consider a typical classification problem. We have a collection of ℓ data points $\{X^i, y^i\}_{i=1}^{\ell}$ where $X^i \equiv (1, x_1^i, x_2^i, ...)$ is a collection of dependent variables (features) and y^i is a categorical independent variable (label) that takes values 0 and 1. The goal is to construct a dashed line that separates the known examples of the two types.

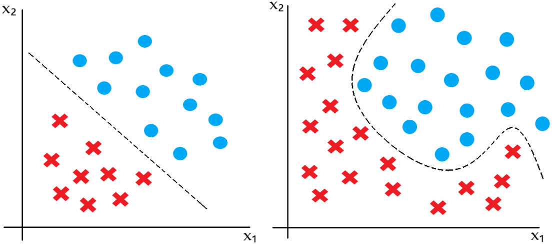

Figure 6. Examples of binary classification.

We restrict attention to one technique – logistic regression – which is simple, general and can be conveniently combined with our deep learning analysis

A hypothesis

As a first step, we form a hypothesis about the functional form of the separating line. For the left panel, it is sufficient to assume that the separating line is linear

$$H_0 : \theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0,$$

but for the right panel, we must use a sufficiently flexible nonlinear separating function such as a higher-order polynomial function,

$$H_0 : \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + ... = 0,$$

where $(\theta_0, \theta_1, ...) \equiv \theta$ are the polynomial coefficients. When $X\theta \equiv \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... > 0$, we conclude that y belongs to class 1 and otherwise, we conclude that it is from class 0.

Estimation

- Our next step is to estimate θ coefficients. Since y is a categorical variable $y \in \{0, 1\}$, we cannot use ordinary least-squares estimator, i.e., we cannot regress y on $X\theta$. Instead, we form a logistic regression
$$H_0 : \log \frac{p}{1-p} = X\theta,$$
where p is the probability that a data point with characteristics $X \equiv \left(1, x_1, x_2, x_1^2, ...\right)$ belongs to class 1, and $\theta \equiv (\theta_0, \theta_1, ..., \theta_m, ..., \theta_M)$ is a coefficient vector.
- The logistic function is an excellent choice for approximating probability:
    - First, it ensures that $p = \frac{1}{1+\exp(-X\theta)} \in (0, 1)$ for any θ and X, and hence p and $(1-p)$ can be interpreted as probabilities that a data point belongs to classes 1 and 0, respectively.
    - Second, $p = \frac{1}{2}$ corresponds to the separation line $X\theta = 0$. Hence, when $p > \frac{1}{2}$, the data point is "above" the separating line $X\theta$, and thus, belongs to the class **1** and if $p < \frac{1}{2}$, the opposite is true.
    - Finally, when $X\theta \to -\infty$ and $X\theta \to +\infty$, we have that $p \to 0$ and $p \to 1$, respectively.

Probability of an observation

The logistic regression provides a convenient way to estimate the decision boundary coefficients θ by using a maximum likelihood estimator. A probability that the data point i belongs to classes 0 and 1 can be represented with a single formula by

$$\text{Prob}\left(y \mid X; \theta\right) = p^y \left(1 - p\right)^{1-y}.$$

Indeed, if $y = 1$, we have $\text{Prob}(y = 1 \mid X; \theta) = \left(p\right)^1 \left(1 - p\right)^0 = p$; and if $y = 0$, we have $\text{Prob}(y = 0 \mid X; \theta) = \left(p\right)^0 \left(1 - p\right)^1 = 1 - p$.

Likelihood function

We search for the coefficient vector θ that maximizes the (log)likelihood of the event such that a given matrix of features $\left\{X^i\right\}_{i=1}^{\ell}$ produces the given output realizations $\left\{y^i\right\}_{i=1}^{\ell}$, i.e.,

$$\max_{\theta} \ln L(\theta) = \ln \prod_{i=1}^{\ell} \left(p\left(X^i;\theta\right)\right)^{y^i} \left(1 - p\left(X^i;\theta\right)\right)^{1-y^i} =$$

$$\sum_{i=1}^{\ell} \left[y^i \ln\left(p\left(X^i;\theta\right)\right) + \left(1-y^i\right) \ln\left(1 - p\left(X^i;\theta\right)\right)\right],$$

where the probability $p\left(X^i;\theta\right) \equiv \frac{1}{1+\exp(-X^i\theta)}$ is given by a logistic function.

Constructing a maximizer

To find the maximizer, we compute the first-order conditions with respect to all coefficients θ_m for $m = 0, ..., M$,

$$\frac{\partial \ln L(\theta)}{\partial \theta_m} = \sum_{i=1}^{\ell} \left[\frac{y^i}{p(X^i;\theta)} \frac{\partial p\left(X^i;\theta\right)}{\partial \theta_m} - \frac{\left(1-y^i\right)}{\left(1-p\left(X^i;\theta\right)\right)} \frac{\partial p\left(X^i;\theta\right)}{\partial \theta_m} \right]$$

$$= \sum_{i=1}^{\ell} \left[y^i x_m^i \left(1 - p\left(X^i;\theta\right)\right) - \left(1-y^i\right) x_m^i p\left(X^i;\theta\right) \right]$$

$$= \sum_{i=1}^{\ell} \left[y^i - p\left(X^i;\theta\right) \right] x_m^i,$$

where x_m^i is the feature m of agent i.

The constructed gradient $\nabla \ln L_\theta(\theta) \equiv \left[\frac{\partial \ln L(\theta)}{\partial \theta_1}, ..., \frac{\partial \ln L(\theta)}{\partial \theta_M} \right]'$ can be used for implementing the gradient descent-style method $\theta \leftarrow \theta - \lambda \nabla \ln L_\theta(\theta)$.

Decisions in divisible versus indivisible labor

- In the divisible labor model, we construct a policy function that determines the hours worked $\frac{n_t^i}{L}$.
- In the indivisible labor model studied here, we construct a decision boundary $\varphi\left(s_t^i;\theta\right)=0$ that separates the employment and unemployment choices conditional on state $s_t^i \equiv \left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t\right)$.
- Whenever $\varphi\left(s_t^i;\theta\right) \geq 0$, the agent is employed $n_t^i=\overline{n}$ and otherwise, the agent is unemployed $n_t^i=0$.
- Let us show how such a decision boundary can be constructed by using the logistic regression classification method.

Decisions in divisible versus indivisible labor

- Since our model has a large number of explanatory variables (state variables) as well as a highly nonlinear decision boundary, we use neural networks for approximating such boundary (instead of the polynomial function).
- We estimate the coefficients of the neural network (weights and biases) by formulating a logistic regression,

$$H_0 : \log \frac{p}{1-p} = \varphi\left(s; \theta\right).$$

- We parameterize the decision functions p_t^i and $\frac{c_t^i}{w_t^i}$ by a sigmoid function in the indivisible labor model:

$$\sigma\left(\zeta_0 + \varphi\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t; \theta\right)\right),$$

where $\varphi\left(\cdot\right)$ is a multilayer neural network parameterized by a vector of coefficients θ (weights and biases), $\sigma\left(z\right) = \frac{1}{1+e^{-z}}$ is a sigmoid function which ensures that $\frac{c_t^i}{w_t^i}$ and p_t^i are bounded in the interval $[0, 1]$, respectively, and ζ_0 is a constant term. (Here, we also parameterize the Lagrange multiplier.

Decisions in divisible versus indivisible labor

- The function p_t^i, allows us to infer the indivisible labor choice directly, specifically, an agent is employed $n_t^i = \overline{n}$ whenever $p_t^i \geq \frac{1}{2}$ and is unemployed otherwise $n_t^i = 0$.
- We can then compute $h_t = \sum_{i=1}^{\ell} v_t^i n^i$ and find W_t and R_t restore the remaining individual and aggregate variables.
- Our next goal is to check if the constructed labor choices are consistent with the individual optimality conditions.
- We use the decision functions p_t^i, $\frac{c_t^i}{w_t^i}$ and μ_t^i to restore the value functions for the employed and unemployed agents $V^E\left(s_t^i;\theta^E\right)$ and $V^U\left(s_t^i;\theta^U\right)$.
- We next construct the labor choice $\widehat{n}_t^i$ implied by these two value functions

$$\widehat{n}_t^i = \begin{cases} \overline{n} \text{ if } V^E = \max\left\{V^E, V^U\right\}, \\ 0 \text{ otherwise.} \end{cases}$$

Decisions in divisible versus indivisible labor

In the solution, the labor choice $\widehat{n}_t^i$ implied by the value functions must coincide with the labor choice n_t^i produced by our decision function for all i and t. If this is not the case, we proceed with training our classifier. To this purpose, we construct the categorical variable $y_t^i \in \{0, 1\}$ such that

$$y_t^i = \begin{cases} 1 \text{ if } \widehat{n}_t^i = \overline{n}, \\ 0 \text{ otherwise,} \end{cases}$$

and we use it to form the (log)likelihood function

$$\ln L(\theta) = \frac{1}{\ell} \sum_{i=1}^{\ell} \left[y_t^i \ln \left(p\left(s_t^i; \theta \right) \right) + \left(1 - y_t^i \right) \ln \left(1 - p\left(s_t^i; \theta \right) \right) \right].$$

We then maximize the likelihood function by using a conventional / stochastic / batch stochastic gradient descent methods. We iterate on the decision functions p_t^i, $\frac{c_t^i}{w_t^i}$ and μ_t^i until convergence.

Implementation difference in construction of divisible and indivisible labor.

- There is an important implementation difference in the construction of the labor choice in the divisible and indivisible labor models.
- In the former model, the optimal labor choice must satisfy FOC and hence, it can be constructed by considering just the current period variables.
- However, this is not true for the indivisible labor model in which the agent chooses to be employed or unemployed depending on which of the two continuation values is larger V^E or V^U.

Prescott et al. (2009): intensive and extensive margins

- Prescott et al. (2009) propose a cleaver approach to modeling the indivisible labor choice under which such a choice can be constructed from the current state variables without the need of constructing value functions.
- They allow for intensive and extensive margins by "discretizing" the FOC. To be specific, they assume that the labor choice is divisible as long as it is above a given threshold $\overline{n}_f$ but it jumps to zero whenever the labor choice falls below $\overline{n}_f$ (i.e., the agent becomes unemployed):

$$\widehat{n}_t^i = \begin{cases} L - \left[\frac{c_i^{-\gamma} W_t \exp\left(v_t^i\right)}{B}\right]^{-1/\eta} \geq \overline{n}_f, \\ 0 \text{ otherwise.} \end{cases}$$

Determining indivisible labor: value functions versus "discretized" FOC

- We borrow from Prescott et al. (2009) the idea of discretizing the FOCs of the divisible labor model, however, we go a step further and we make the labor choice entirely indivisible by assuming that n_t^i can take just two values 0 (unemployed) and $\overline{n}$ (employed):

$$\widehat{n}_t^i = \begin{cases} \overline{n} \text{ if } L - \left[\frac{c_i^{-\gamma} W_t \exp\left(v_t^i\right)}{B} \right]^{-1/\eta} \geq \overline{n}_f, \\ 0 \text{ otherwise.} \end{cases}$$

- The above approach can be a simple and effective alternative to conventional methods that solve for indivisible labor by constructing the value functions V^E and V^U explicitely.

Algorithm 2: Deep learning for indivisible labor model

Algorithm 2: Deep learning for the indivisible labor model.
Step 0: (Initialization).
Construct initial state $\left(\left\{k_0^i, v_0^i\right\}_{i=1}^{\ell}, z_0\right)$ and parameterize the decision functions by $$\left\{p_t^i, \frac{c_t^i}{w_t^i}\right\} = \sigma\left(\zeta_0 + \varphi\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t; \theta\right)\right),$$ $$\mu_t^i = \exp\left(\zeta_0 + \varphi\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t; \theta\right)\right),$$ where p_t^i is the probability of being employed.

Algorithm 2: Deep learning for indivisible labor model (cont.)

Algorithm 2: Deep learning for the indivisible labor model.

Step 1: (Evaluation of decision functions).

Given $\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t\right)$, compute $n_t^i = \overline{n}$ if $p_t^i \geq \frac{1}{2}$ and $n_t^i = 0$ if $p_t^i < \frac{1}{2}$.

Compute w_t^i and $\frac{c_t^i}{w_t^i}$,and find R_t and and W_t; and find k_{t+1}^i from the budget constraint for all agents $i = 1, ..., \ell$.

Option 1: Construct V^E and V^U and find $\widehat{n}_t^i = \begin{cases} \overline{n} \text{ if } V^E = \max\left\{V^E, V^U\right\}, \\ 0 \text{ otherwise.} \end{cases}$

Option 2: Use the discretized FOC $\widehat{n}_t^i = \begin{cases} \overline{n} \text{ if } L - \left[\frac{c_i^{-\gamma} W_t \exp\left(v_t^i\right)}{B}\right]^{-1/\eta} \geq \overline{n}_f, \\ 0 \text{ otherwise.} \end{cases}$

Define $y_t^i = \begin{cases} 1 \text{ if } \widehat{n}_t^i = \overline{n}, \\ 0 \text{ otherwise,} \end{cases}$ for each s_t^i.

Algorithm 2: Deep learning for divisible labor model (cont.)

Algorithm 2: Deep learning for the indivisible labor model.
Step 2: (Construction of Euler residuals).
Draw two random sets of individual productivity shocks $\Sigma_1 = \left(\epsilon_1^1, ..., \epsilon_1^\ell\right)$, $\Sigma_2 = \left(\epsilon_2^1, ..., \epsilon_2^\ell\right)$ and two aggregate shocks $\epsilon_{1,}$, ϵ_2, to construct $$\Xi(\theta) = \left\{ \left[\Psi^{FB}\left(1 - \frac{c_t^i}{w_t^i}, 1 - \mu_t^i\right)\right]^2 + \varpi_n \left[y_t^i \ln\left(p\left(s_t^i;\theta\right)\right) + \left(1 - y_t^i\right)\ln\left(1 - p\left(s_t^i;\theta\right)\right)\right]^2 + \varpi_\mu \left[\frac{\beta\left[\left(c_{t+1}^i\right)^{-\gamma} R_{t+1} \middle\| \Sigma'_{t+1}, \epsilon'_{t+1}\right]}{\left(c_t^i\right)^{-\gamma}} - \mu_t^i\right]\left[\frac{\beta\left[\left(c_{t+1}^i\right)^{-\gamma} R_{t+1} \middle\| \Sigma''_{t+1}, \epsilon''_{t+1}\right]}{\left(c_t^i\right)^{-\gamma}} - \mu_t^i\right] \right\},$$ where $\Psi^{FB}(a,b) = a + b - \sqrt{a^2+b^2}$ is a Fischer-Burmeister function; and ϖ_n, ϖ_μ are given weights.
Step 3: (Training).
...
Step 4: (Simulation).

Training errors and running time

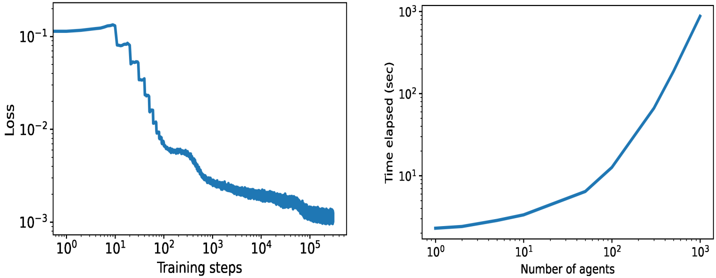

Figure 7. Training errors and running time for indivisible labor model.

The solution for divisible labor model

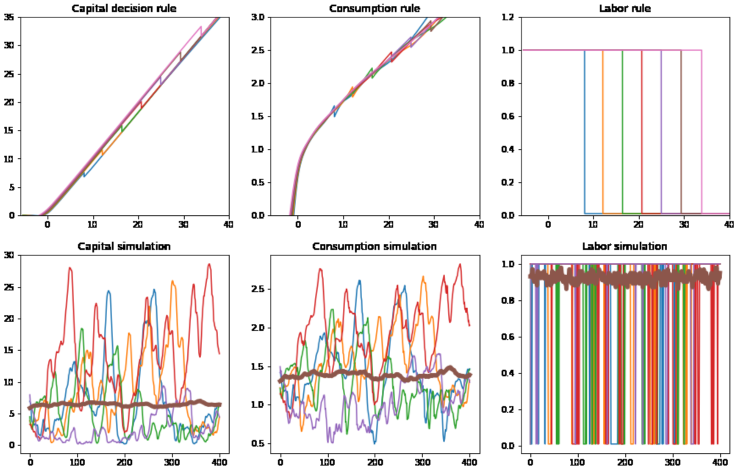

Figure 8. Solution to indivisible labor model under $\gamma = 1$ and $\eta = 1$.

Deep learning method for the model with 3 states

Multiclass classification problem

We again have a collection of ℓ data points $\{X^i, y^i\}_{i=1}^{\ell}$ where $X^i \equiv (1, x_1^i, x_2^i, ...)$ is composed of dependent variables (features) but now y^i is a categorical independent variable (label) that takes K values. Our goal is to construct the lines that separate the classes 1, 2 and 3.

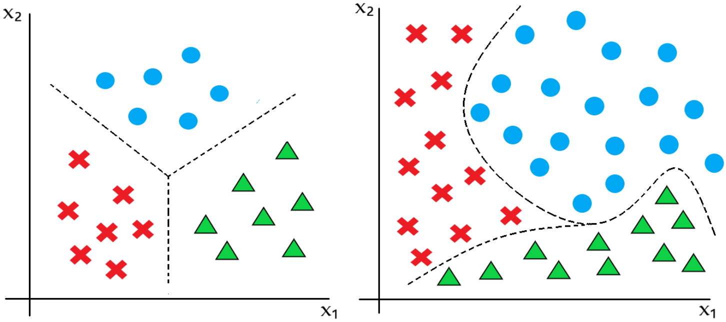

Figure 9. Examples of multiclass classification.

From multiclass to binary classification problem

- A popular approach in machine learning is to reformulate a multiclass classification problem as a collection of binary classification problems.
- The key assumption behind this approach is the hypothesis of an independence of irrelevant alternatives.
- In our analysis, that means that the choice between $\{\times\}$ and $\{\triangle\}$ is independent of the availability of $\{\mathbf{o}\}$, the choice between $\{\triangle\}$ and $\{\mathbf{o}\}$ is independent of the availability of $\{\times\}$ and the choice between $\{\mathbf{o}\}$ and $\{\times\}$ is independent of the availability of $\{\triangle\}$.
- Two binary reformulations of a multiclass classification problems are the *one-versus-one* and *one-versus-rest* (or *one-versus-all*) classifiers,

$$\ln \frac{p(\times)}{p(\mathbf{o})} = X\theta^{(1)} \quad \ln \frac{p(\triangle)}{p(\mathbf{o})} = X\theta^{(2)} \quad \ln \frac{p(\triangle)}{p(\times)} = X\theta^{(3)},$$

$$\ln \frac{p(\times)}{p(\mathbf{o})+p(\triangle)} = X\theta^{(1)} \quad \ln \frac{p(\triangle)}{p(\mathbf{o})+p(\times)} = X\theta^{(2)} \quad \ln \frac{p(\mathbf{o})}{p(\triangle)+p(\times)} = X\theta^{(3)},$$

where $\theta^{(1)}$, $\theta^{(2)}$ and $\theta^{(3)}$ are the regression coefficients and X is the matrix of features.

Training multi class classifiers

- To train the constructed multiclass classifiers, we may omit one of three regressions by imposing the restriction that the probabilities are added to one.
- For the one-versus-one classifier, the first two regressions imply $p(\times) = p(\mathbf{o}) \exp\left(X\theta^{(1)}\right)$ and $p(\triangle) = p(\mathbf{o}) \exp\left(X\theta^{(2)}\right)$ so that $p(\mathbf{o})\left(1 + \exp\left(X\theta^{(1)}\right) + \exp\left(X\theta^{(2)}\right)\right) = 1$.
- In turn, for the one-versus-rest classifier, in the first regression, we replace $p(\mathbf{o}) + p(\triangle)$ with $1 - p(\times)$ and in the second regression, we replace $p(\mathbf{o}) + p(\times)$ with $1 - p(\triangle)$.

Consequently, we can re-write two classifiers as

$$p(\times) = \exp\left(X\theta^{(1)}\right) p(o), \quad p(\triangle) = \exp\left(X\theta^{(2)}\right) p(\mathbf{o}) \quad ,$$
$$p(\mathbf{o}) = \frac{1}{1+\exp\left(X\theta^{(1)}\right)+\exp\left(X\theta^{(2)}\right)},$$

$$p(\times) = \frac{1}{1+\exp(-X\theta^{(1)})} \quad p(\triangle) = \frac{1}{1+\exp(-X\theta^{(2)})} \quad p(\mathbf{o}) = 1 - p(\times) - p(\triangle).$$

Symmetric one-versus-rest classifier

- Note that in the above expressions, we treat the normalizing class $\{\mathbf{o}\}$ differently from the other two classes $\{\triangle, \times\}$.
- There is also a symmetric version of the *one-versus-rest* method in which all K classes are treated identically by estimating K unnormalized one-versus-rest logistic regressions $\ln p\,(\times) = X\theta^{(1)}$, $\ln p\,(\triangle) = X\theta^{(2)}$, $\ln p\,(\mathbf{o}) = X\theta^{(3)}$ and by normalizing the exponential function ex-post by their sum.
- This classifier is called softmax and it is a generalization of a logistic function to multiple dimensions,

$$p\,(\times) = \tfrac{1}{\Sigma} \exp\left(X\theta^{(1)}\right)$$
$$p\,(\triangle) = \tfrac{1}{\Sigma} \exp\left(X\theta^{(2)}\right) \quad ,$$
$$p\,(\mathbf{o}) = \tfrac{1}{\Sigma} \exp\left(X\theta^{(3)}\right),$$

  where $\Sigma = \exp\left(X\theta^{(1)}\right) + \exp\left(X\theta^{(2)}\right) + \exp\left(X\theta^{(3)}\right)$.
- The symmetric treatment is convenient in deep learning analysis because it allows us to use a neural network with K symmetric outputs.

Likelihood function for softmax classifier

The log-likelihood function for the softmx classifier is similar to the one for the binary classifier except that we also do a summation over K of possible outcomes,

$$\max_{\theta_1,\ldots,\theta_K} \ln L\left(\theta_1, \ldots, \theta_K\right)$$

$$= \frac{1}{K\ell} \sum_{k=1}^{K} \sum_{i=1}^{\ell} \left[y^{i,k} \ln \left(p \left(X^i; \theta^k \right) \right) + \left(1 - y^{i,k}\right) \ln \left(1 - p \left(X^i; \theta^k \right) \right) \right],$$

where $y^{i,k}$ is a categorical variable constructed so that $y^{i,k} = 1$ if observation i belongs to class k and it is zero otherwise. Again, we maximize the constructed likelihood function by using a gradient descent style method, $\theta \leftarrow \theta - \lambda \nabla \ln L_\theta\left(\theta\right)$.

Discrete choice in three state model

- We next extend our indivisible labor heterogeneous-agent model with two employment choices $\{0, \overline{n}\}$ to three employment choices $\{0, \underline{n}, \overline{n}\}$.
- We parameterize not one but three decision boundaries that separate the three employment choices, so we use a sigmoid function to parameterize four functions $\frac{p_t^i(\overline{n})}{\Sigma}$, $\frac{p_t^i(\underline{n})}{\Sigma}$, $\frac{p_t^i(0)}{\Sigma}$, $\frac{c_t^i}{w_t^i}$, specifically:

$$\sigma\left(\zeta_0 + \varphi\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t; \theta\right)\right),$$

where $\varphi(\cdot)$ is a multilayer neural network parameterized by a vector of coefficients θ (weights and biases), $\Sigma \equiv p_t^i(\overline{n}) + p_t^i(\underline{n}) + p_t^i(0)$ normalizes the probabilities to one; $\sigma(z) = \frac{1}{1+e^{-z}}$ is a sigmoid function which ensures that $\frac{c_t^i}{w_t^i}$ and $\frac{p_t^i(\overline{n})}{\Sigma}$, $\frac{p_t^i(\underline{n})}{\Sigma}$ and $\frac{p_t^i(0)}{\Sigma}$ are bounded in the interval $[0, 1]$, and ζ_0 is a constant term. (In addition, we also parameterize the Lagrange multiplier).

Verifying the optimality conditions

- Our next goal is to check if the constructed labor choices are consistent with the individual optimality conditions.
- To validate the individual choices, we use the decision functions $\frac{p_t^i(\overline{n})}{\Sigma}$, $\frac{p_t^i(\underline{n})}{\Sigma}$, $\frac{p_t^i(0)}{\Sigma}$, $\frac{c_t^i}{w_t^i}$ and μ_t^i to recover the value functions for employed, part-time employed and unemployed agents, V^E, V^{PT} and V^U, respectively, using the appropriately formulated Bellman equations; see Chang and Kim (2007).
- We then construct the labor choice $\widehat{n}_t^i$ implied by such value functions,

$$\widehat{n}_t^i = \begin{cases} \overline{n} \text{ if } V^E = \max\left\{V^E,\ V^{PT}, V^U\right\}, \\ \underline{n} \text{ if } V^{PT} = \max\left\{V^E,\ V^{FT}, V^U\right\}, \\ 0 \text{ otherwise.} \end{cases}$$

- In the solution, the labor choice implied by the value function $\widehat{n}_t^i$ must coincide with the labor choice produced by our decision function n_t^i for all i, t.
- If this is not the case, we proceed to training of our classifier.

Training the model

- To this purpose, we construct the categorical variable $y_t^i \equiv \left(y_t^{i,1}, y_t^{i,2}, y_t^{i,3}\right)$ such that

$$y_t^i = \begin{cases} (1,0,0) & \text{if } \widehat{n}_t^i = \overline{n}, \\ (0,1,0) & \text{if } \widehat{n}_t^i = \underline{n}, \\ (0,0,1) & \text{otherwise.} \end{cases}$$

- We then formulate the (log)likelihood function

$$\ln L\left(\theta^{(1)}, \theta^{(2)}, \theta^{(3)}\right)$$
$$= \frac{1}{3\ell} \sum_{k=1}^{3} \sum_{i=1}^{\ell} \left[\widehat{y}_t^{i,k} \ln\left(p\left(s_t^i; \theta^{(k)}\right)\right) + \left(1 - \widehat{y}_t^{i,k}\right) \ln\left(1 - p\left(s_t^i; \theta^{(k)}\right)\right)\right].$$

- We train the model to maximize the likelihood function by using a conventional / stochastic / batch stochastic gradient descent method.
- We iterate on the decision functions $p_t^i(\overline{n}), p_t^i(\underline{n}), p_t^i(0)$, $\frac{c_t^i}{w_t^i}$ and μ_t^i until convergence.

Determining three-state labor: value functions versus "discretized" FOC

- Chang and Kim (2007) consider a related heterogeneous-agent model with three states but they allow for intensive and extensive margins.
- In contrast, we assume an entirely discrete choice between the three employment states:

$$\widehat{n}_t^i = \begin{cases} \overline{n} \text{ if } L - \left[\frac{c_i^{-\gamma} W_t \exp\left(v_t^i\right)}{B}\right]^{-1/\eta} \geq \overline{n}_f \\ \underline{n} \text{ if } L - \left[\frac{c_i^{-\gamma} W_t \exp\left(v_t^i\right)}{B}\right]^{-1/\eta} \in [\overline{n}_p, \overline{n}_f] \\ 0 \text{ otherwise} \end{cases}$$

- Thus, we assume that the agent chooses full-time employment, $n_t^i = \overline{n}$, whenever her labor choices implied by the FOC of the divisible labor mode is above a threshold $\overline{n}_f$; she chooses part-time employment, $n_t^i = \underline{n}$, whenever it belongs to the interval $[\overline{n}_p, \overline{n}_f]$; and she chooses unemployment whenever it falls below the part-time employment threshold $\overline{n}_p$.

Algorithm 3: Deep learning for model with full and part-time employment

Algorithm 3: Deep learning for the model with full and partial employment.
Step 0: (Initialization).
Construct initial state $\left(\left\{k_0^i, v_0^i\right\}_{i=1}^{\ell}, z_0\right)$ and parameterize the decision functions by $\left\{\frac{p_t^i(\overline{n})}{\Sigma}, \frac{p_t^i(\underline{n})}{\Sigma}, \frac{p_t^i(0)}{\Sigma}, \frac{c_t^i}{w_t^i}\right\} = \sigma\left(\zeta_0 + \varphi\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t; \theta\right)\right)$, where $p_t^i\left(\overline{n}\right)$, $p_t^i\left(\underline{n}\right)$ and $p_t^i\left(0\right)$ are the probabilities to be full- and part-time employed and unemployed, respectively; and $\Sigma \equiv p_t^i\left(\overline{n}\right) + p_t^i\left(\underline{n}\right) + p_t^i\left(0\right)$ is a normalization of probability to one.

Algorithm 3: Deep learning for model with full and part-time employment (cont.)

Algorithm 3: Deep learning for the model with full and partial employment.

Step 1: (Evaluation of decision functions).

Given state $\left(k_t^i, v_t^i, \left\{k_t^i, v_t^i\right\}_{i=1}^{\ell}, z_t\right)$, set $n_t^i = \overline{n}$, $n_t^i = \underline{n}$ and $n_t^i = 0$ depending o
on which probability $p_t^i\left(\overline{n}\right)$, $p_t^i\left(\underline{n}\right)$ and $p_t^i\left(0\right)$ is the largest. Compute w_t^i, $\frac{c_t^i}{w_t^i}$ from
the decision rules and find k_{t+1}^i from the budget constraint for all agents $i = 1, ...\ell$.
Reconstruct V^E, V^{PT} and V^U, respectively.

Find $\widehat{n}_t^i = \begin{cases} \overline{n} \text{ if } V^E = \max\left\{V^E,\ V^{PT}, V^U\right\}, \\ \underline{n} \text{ if } V^{PT} = \max\left\{V^E,\ V^{FT}, V^U\right\}, \\ 0 \text{ otherwise.} \end{cases}$

and define $y_t^i = \begin{cases} (1,0,0) & \text{if } \widehat{n}_t^i = \overline{n}, \\ (0,1,0) & \text{if } \widehat{n}_t^i = \underline{n}, \\ (0,0,1) & \text{otherwise.} \end{cases}$ for each s_t^i.

Algorithm 3: Deep learning for model with full and part-time employment (cont)

Algorithm 3: Deep learning for the model with full and partial employment.
Option 1: Construct V^E, V^{PT}, V^U and $\widehat{n}_t^i = \begin{cases} \overline{n} \text{ if } V^E = \max\left\{V^E,\ V^{PT}, V^U \right. \\ \underline{n} \text{ if } V^{PT} = \max\left\{V^E,\ V^{FT}, V \right. \\ 0 \text{ otherwise.} \end{cases}$
Option 2: From discretized FOC $\widehat{n}_t^i = \begin{cases} \overline{n} \text{ if } L - \left[\frac{c_i^{-\gamma} W_t \exp\left(v_t^i\right)}{B}\right]^{-1/\eta} \geq \overline{n}_f \\ \underline{n} \text{ if } L - \left[\frac{c_i^{-\gamma} W_t \exp\left(v_t^i\right)}{B}\right]^{-1/\eta} \in [\overline{n}_p, \overline{n}_f \\ 0 \text{ otherwise} \end{cases}$
Define $y_t^i = \begin{cases} (1,0,0) & \text{if } \widehat{n}_t^i = \overline{n}, \\ (0,1,0) & \text{if } \widehat{n}_t^i = \underline{n}, \\ (0,0,1) & \text{otherwise.} \end{cases}$ for each s_t^i.

Algorithm 3: Deep learning for model with full and part-time employment (cont)

Algorithm 3: Deep learning for the model with full and partial employment.
Step 2: (Construction of Euler residuals).
Draw two random sets of individual productivity shocks $\Sigma_1 = \left(\epsilon_1^1, ..., \epsilon_1^\ell\right)$, $\Sigma_2 = \left(\epsilon_2^1, ..., \epsilon_2^\ell\right)$ and two aggregate shocks $\epsilon_{1,}$, ϵ_2, and construct the residuals $$\Xi(\theta) = \left\{ \left[\Psi^{FB} \left(1 - \frac{c_t^i}{w_t^i}, 1 - \mu_t^i \right) \right]^2 + \frac{\varpi_n}{3} \sum_{k=1}^{3} \left[\widehat{y}_t^{i,k} \ln \left(p \left(s_t^i; \theta^{(k)} \right) \right) + \left(1 - \widehat{y}_t^{i,k} \right) \ln \left(1 - p \left(s_t^i; \theta^{(k)} \right) \right) \right]^2 + \varpi_\mu \left[\frac{\beta \left[\left(c_{t+1}^i \right)^{-\gamma} R_{t+1} \middle\vert \Sigma'_{t+1}, \epsilon'_{t+1} \right]}{\left(c_t^i \right)^{-\gamma}} - \mu_t^i \right] \left[\frac{\beta \left[\left(c_{t+1}^i \right)^{-\gamma} R_{t+1} \middle\vert \Sigma''_{t+1}, \epsilon''_{t+1} \right]}{\left(c_t^i \right)^{-\gamma}} - \mu_t^i \right] \right\},$$ where $\Psi^{FB}(a, b) = a + b - \sqrt{a^2 + b^2}$ is a Fischer-Burmeister function; and ϖ_n, ϖ_μ are given weights.
Step 3: (Training).
... Step 4: (Simulation).

Training errors and running time

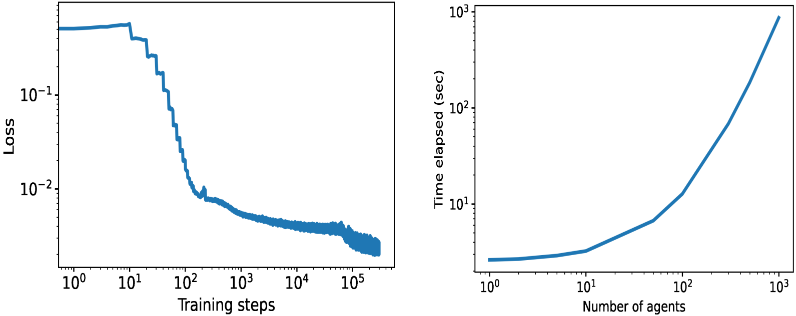

Figure 10. Training errors and running time for three-state employment m

The solution for divisible labor model

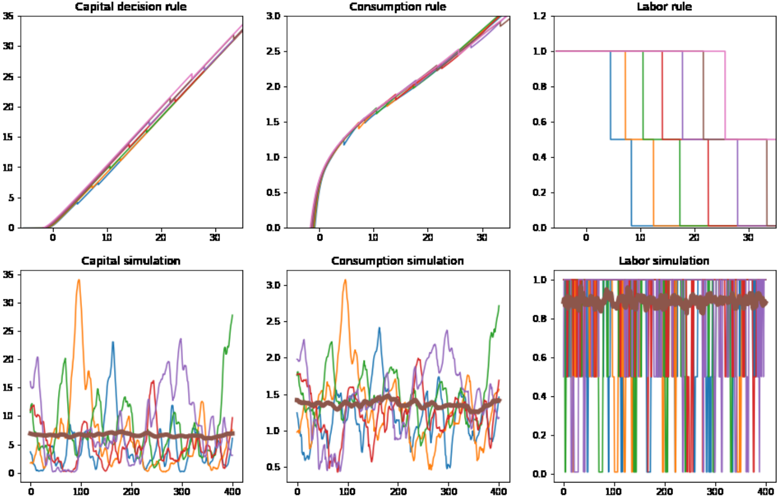

Figure 11. Solution to the three-state employment model.

Conclusion

- This paper shows how to use deep learning classification approach borrowed from data science for modeling discrete choices in dynamic economic models.
- A combination of the state-of-the-art machine learning techniques makes the proposed method tractable in problems with very high dimensionality – hundreds and even thousands of heterogeneous agents.
- We investigate just one example – discrete labor choice – but the proposed deep learning classification method has a variety of potential applications such as sovereign default models, models with retirement, and models with indivisible commodities, in particular, housing.