

# Deep Learning Classification: Modeling Discrete Labor Choice

Lilia Maliar, Serguei Maliar

Turin, August 28, 2025

ESOMAS Department, University of Turin

# Discrete- versus continuous-set choices

Macroeconomic models are mostly built on **continuous**-set choices (how to distribute time, wealth in any proportion).

But certain economic choices are **discrete** (retire or not, buy house or not, default or not, work or not)



Trends in Cognitive Sciences



**The progress in modeling discrete choices is still limited!**

# What makes modelling discrete choice hard

- Dynamic discrete choice models often fall under the class of *NP-hard* in terms of computational complexity.
- When discrete choice is embedded in a dynamic optimization problem, the agent must compute value functions for each discrete option over the entire state space.
- This (might) lead to:
  - combinatorial explosion: the number of possible histories or paths grows exponentially.
  - non-differentiability: discrete choices make it hard to use standard calculus-based optimization.
  - Bellman equations become piecewise and hard to solve with continuous methods.

## Our analysis: why deep learning helps

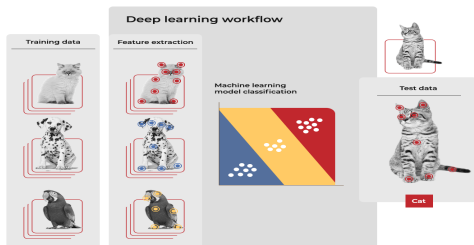
- We introduce a deep learning classification (DLC) method to:
  - avoid explicit enumeration over all discrete options.
  - approximate optimal choices using probabilistic classifiers (e.g., softmax).
  - leverage gradient descent rather than brute-force combinatorics.
- This doesn't change the theoretical hardness, but provides a practical **workaround** using function approximation.

# Our results

- We use DLC to solve 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 birds.
- 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 (equivalent to maximizing the likelihood function).
- See Goodfellow, Bengio and Courville (2016) for a survey of classification methods in data science.

# Classification method for discrete choice in economics

- Our example of using classification method in macroeconomics:
  - we 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 under which such choices are optimal.
- **Remark:** Earlier literature on indivisible labor (Rogerson (1994) and Hansen (1993)) constructs discrete choice by introducing lotteries.
  - Our probabilities have a different meaning: they indicate which discrete choice is most likely to be optimal and hence, is selected.

# Krusell and Smith's (1999) model

- Deep learning 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 intractable with conventional solution methods.
- To illustrate this power, we solve Krusell and Smith's (1998) model in which the agents face indivisible labor choices, specifically:
  - *a version model with continuous choices (i.e., divisible labor);*
  - *an indivisible-labor version with 2 discrete employment states (employed and unemployed);*
  - *an indivisible-labor version with 3 discrete employment states (employed, unemployed and part-time employed agent).*

# The model

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

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

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

## 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 labor,  $h_t = \sum_{i=1}^{\ell} v_t^i n_t^i$  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.

# Three versions of the model

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

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

## Deep learning method for the divisible labor model

## Deep learning method for the 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, Maliar and Winant (JME 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.

## Divisible labor model

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

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

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

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

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

where  $v_t^i$  is an idiosyncratic labor productivity shock.

# Training errors and running time

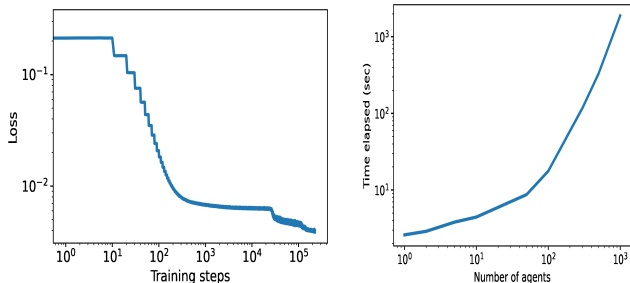


Figure 1. Training errors and running time for the divisible labor model.

# The solution for the divisible labor model

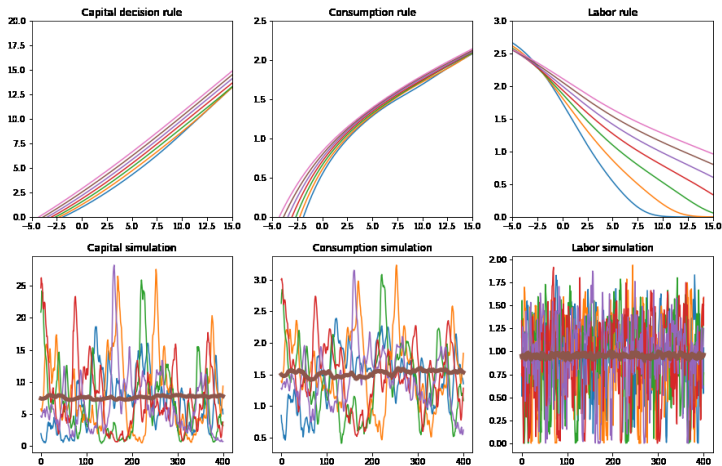


Figure 2. Solution to the divisible labor model.

## Deep learning method for the indivisible labor model

## Indivisible labor model with 2 employment states

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

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

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

## Logistic regression

Let us consider a typical classification problem. We have  $\ell$  data points  $\{X^i, y^i\}_{i=1}^{\ell}$  where  $X^i \equiv (1, x_1^i, x_2^i, \dots)$  is a collection of explanatory variables (features) and  $y^i$  is a categorical dependent 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.

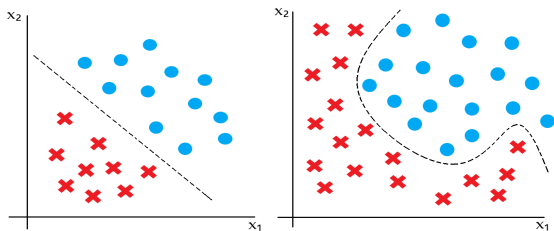


Figure 3. Examples of binary classification.

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,

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

where  $(\theta_0, \theta_1, \dots) \equiv \theta$  are the polynomial coefficients.

- When  $X\theta \equiv \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots > 0$ , we conclude that  $y$  belongs to class 1 and otherwise, we conclude that it is from class 0.

## Estimation of unknown coefficients

- Our next step is to estimate  $\theta$  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  $X \equiv (1, x_1, x_2, x_1^2, \dots)$  belongs to class 1, and  $\theta \equiv (\theta_0, \theta_1, \dots, \theta_m, \dots, \theta_M)$  is a coefficient vector.

- The **sigmoid (logistic) function** is an excellent choice for approximating probability:
  - it ensures that  $p = \frac{1}{1+\exp(-X\theta)} \in (0, 1)$  for any  $\theta$  and  $X$ .
  - $p = \frac{1}{2}$  corresponds to the separation line  $X\theta = 0$ .  $\implies$  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.
  - when  $X\theta \rightarrow -\infty$  and  $X\theta \rightarrow +\infty$ , we have that  $p \rightarrow 0$  and  $p \rightarrow 1$ , respectively.

# Probability of an observation

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

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

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

## Likelihood function

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

$$\begin{aligned}\max_{\theta} \ln \mathcal{L}(\theta) &= \ln \prod_{i=1}^{\ell} (p(X^i; \theta))^{y^i} (1 - p(X^i; \theta))^{1-y^i} = \\ &= \sum_{i=1}^{\ell} [y^i \ln(p(X^i; \theta)) + (1 - y^i) \ln(1 - p(X^i; \theta))],\end{aligned}$$

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

## Constructing a maximizer

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

$$\begin{aligned}\frac{\partial \ln \mathcal{L}(\theta)}{\partial \theta_m} &= \sum_{i=1}^{\ell} \left[ \frac{y^i}{p(X^i; \theta)} \frac{\partial p(X^i; \theta)}{\partial \theta_m} - \frac{(1-y^i)}{(1-p(X^i; \theta))} \frac{\partial p(X^i; \theta)}{\partial \theta_m} \right] \\ &= \sum_{i=1}^{\ell} [y^i x_m^i (1-p(X^i; \theta)) - (1-y^i) x_m^i p(X^i; \theta)] \\ &= \sum_{i=1}^{\ell} [y^i - p(X^i; \theta)] x_m^i,\end{aligned}$$

where  $x_m^i$  is a feature  $m$  of agent  $i$ .

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

## 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  $\varphi(s_t^i; \theta) = 0$  that separates the employment and unemployment choices conditional on state  $s_t^i \equiv \left( k_t^i, v_t^i, \{k_t^i, v_t^i\}_{i=1}^\ell, z_t \right)$ .
- Whenever  $\varphi(s_t^i; \theta) \geq 0$ , the agent is employed  $n_t^i = \bar{n}$  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 the indivisible labor model

- 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  $\varphi(s; \theta)$  (instead of the polynomial function).
- We estimate the coefficients of the neural network (weights and biases) by formulating a logistic regression,

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

- We parameterize the decision function for  $p_t^i$  by a sigmoid function in the indivisible labor model:

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

where  $\varphi(\cdot)$  is a multilayer neural network parameterized by a vector of coefficients  $\theta$  (weights and biases),  $\sigma(z) = \frac{1}{1+e^{-z}}$  is a sigmoid function which ensures that  $p_t^i$  is bounded in the interval  $[0, 1]$ , respectively, and  $\zeta_0$  is a constant term.

## Decisions in the indivisible labor model (cont.)

- The function  $p_t^i$ , allows us to infer the indivisible labor choice directly, specifically, an agent is employed  $n_t^i = \bar{n}$  whenever  $p_t^i \geq \frac{1}{2}$  and is unemployed otherwise  $n_t^i = 0$ .
- We can then compute  $h_t = \sum_{i=1}^{\ell} v_t^i n_t^i$  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 function  $p_t^i$  to restore the value functions for the employed and unemployed agents  $V^E(s_t^i; \theta^E)$  and  $V^U(s_t^i; \theta^U)$ .
- We next construct the labor choice  $\hat{n}_t^i$  implied by these two value functions

$$\hat{n}_t^i = \begin{cases} \bar{n} & \text{if } V^E = \max \{V^E, V^U\}, \\ 0 & \text{otherwise.} \end{cases}$$

## Decisions in the indivisible labor model (cont.)

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

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

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

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

We then maximize the likelihood function by using a conventional / stochastic / batch stochastic gradient descent methods. We iterate on the decision function  $p_t^i$  until convergence.

# Training errors and running time

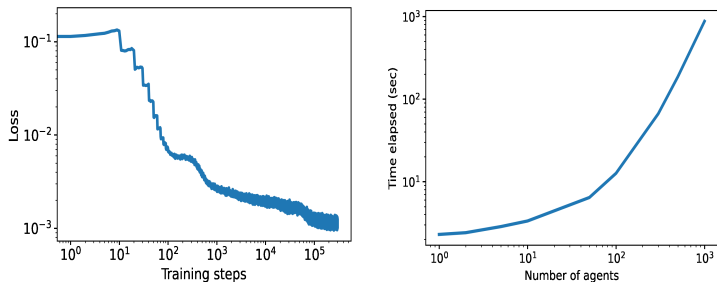


Figure 7. Training errors and running time for the indivisible labor model.

# The solution for the indivisible labor model

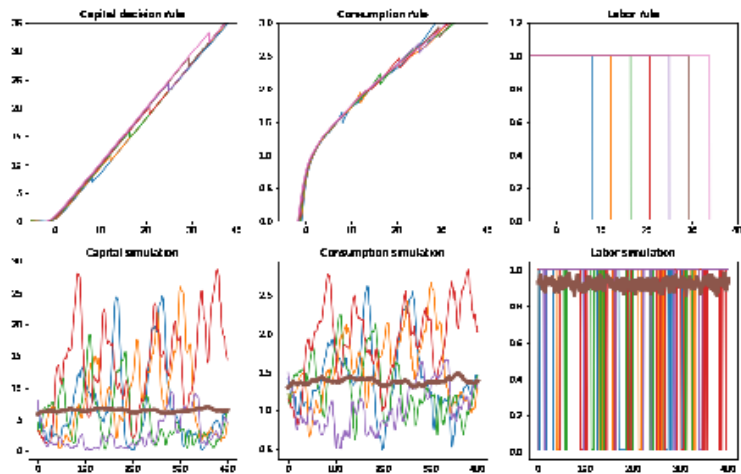


Figure 8. Solution to indivisible labor model under  $\gamma = 1$  and  $\eta = 1$ .

## Deep learning method for the model with 3 employment states

## Indivisible labor model with 3 employment states

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

$$n_t^i = \bar{n} \text{ if } V^{FT} = \max \{V^U, V^{FT}, V^{PT}\}$$

$$n_t^i = \underline{n} \text{ if } V^{PT} = \max \{V^U, V^{FT}, V^{PT}\}$$

$$n_t^i = 0 \text{ otherwise}$$

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

## Multiclass classification problem

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

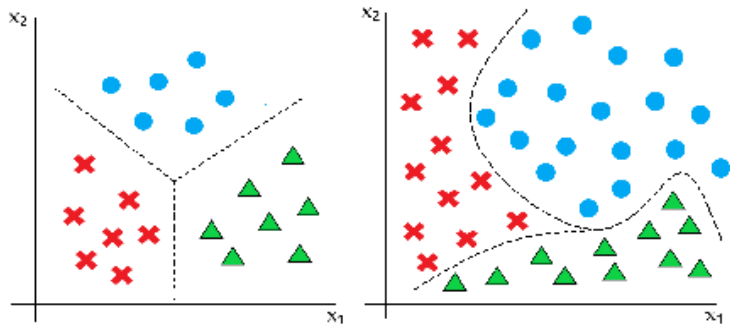


Figure 9. Examples of multiclass classification.

## 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  $\{\circ\}$ , the choice between  $\{\triangle\}$  and  $\{\circ\}$  is independent of the availability of  $\{\times\}$  and the choice between  $\{\circ\}$  and  $\{\times\}$  is independent of the availability of  $\{\triangle\}$ .
- Two binary reformulations of a multiclass classification problems are the *one-versus-one* (o-v-o) and *one-versus-rest* (o-v-r) classifiers,

$$\text{o-v-o: } \ln \frac{p(\times)}{p(\circ)} = X\theta^{(1)} \quad \ln \frac{p(\triangle)}{p(\circ)} = X\theta^{(2)} \quad \ln \frac{p(\triangle)}{p(\times)} = X\theta^{(3)},$$

$$\text{o-v-r: } \ln \frac{p(\times)}{p(\circ)+p(\triangle)} = X\theta^{(1)} \quad \ln \frac{p(\triangle)}{p(\circ)+p(\times)} = X\theta^{(2)} \quad \ln \frac{p(\circ)}{p(\triangle)+p(\times)} = X\theta^{(3)},$$

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

$$\text{o-v-o} : \quad p(\times) = \exp(X\theta^{(1)}) p(\mathbf{o}), \quad p(\Delta) = \exp(X\theta^{(2)}) p(\mathbf{o}), \\ p(\mathbf{o}) = \frac{1}{1 + \exp(X\theta^{(1)}) + \exp(X\theta^{(2)})},$$

$$\text{o-v-r} : \quad p(\times) = \frac{1}{1 + \exp(-X\theta^{(1)})}, \quad p(\Delta) = \frac{1}{1 + \exp(-X\theta^{(2)})}, \\ p(\mathbf{o}) = 1 - p(\times) - p(\Delta).$$

## Symmetric one-versus-rest classifier

- Note that in the above expressions, we treat the normalizing class  $\{\circ\}$  differently from the other two classes  $\{\Delta, \times\}$ .
- 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 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,

$$p(\times) = \frac{1}{\Sigma} \exp\left(X\theta^{(1)}\right) ,$$

$$p(\Delta) = \frac{1}{\Sigma} \exp\left(X\theta^{(2)}\right) ,$$

$$p(\circ) = \frac{1}{\Sigma} \exp\left(X\theta^{(3)}\right) ,$$

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 softmax classifier is similar to the one for the binary classifier except that we also do a summation over  $K$  possible outcomes,

$$\begin{aligned} & \max_{\theta_1, \dots, \theta_K} \ln \mathcal{L}(\theta_1, \dots, \theta_K) \\ &= \frac{1}{K\ell} \sum_{k=1}^K \sum_{i=1}^{\ell} \left[ y^{i,k} \ln \left( p \left( X^i; \theta^k \right) \right) + (1 - y^{i,k}) \ln \left( 1 - p \left( X^i; \theta^k \right) \right) \right], \end{aligned}$$

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,  $\theta \leftarrow \theta - \lambda \nabla \ln \mathcal{L}_{\theta}(\theta)$ .

## Discrete choice in the three state model

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

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

where  $\varphi(\cdot)$  is a multilayer neural network parameterized by a vector of coefficients  $\theta$  (weights and biases),  $\Sigma \equiv p_t^i(\bar{n}) + p_t^i(\underline{n}) + p_t^i(0)$  normalizes the probabilities to one;  $\sigma(z) = \frac{1}{1+e^{-z}}$  is a sigmoid function which ensures that  $\frac{p_t^i(\bar{n})}{\Sigma}$ ,  $\frac{p_t^i(\underline{n})}{\Sigma}$  and  $\frac{p_t^i(0)}{\Sigma}$  are bounded in the interval  $[0, 1]$ , and  $\zeta_0$  is a constant term.

- The decision functions  $\frac{p_t^i(\bar{n})}{\Sigma}$ ,  $\frac{p_t^i(\underline{n})}{\Sigma}$ ,  $\frac{p_t^i(0)}{\Sigma}$  allow us to infer the indivisible labor choice directly, specifically, the employment state  $n_t^i$  is the one corresponding to the highest probability.

## 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  $\frac{p_t^i(\bar{n})}{\Sigma}$ ,  $\frac{p_t^i(n)}{\Sigma}$ ,  $\frac{p_t^i(0)}{\Sigma}$  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.
- We then construct the labor choice  $\hat{n}_t^i$  implied by such value functions,

$$\hat{n}_t^i = \begin{cases} \bar{n} & \text{if } V^E = \max \{V^E, V^{PT}, V^U\}, \\ n & \text{if } V^{PT} = \max \{V^E, V^{PT}, V^U\}, \\ 0 & \text{otherwise.} \end{cases}$$

- In the solution, the labor choice implied by the value function  $\hat{n}_t^i$  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  $y_t^i \equiv (y_t^{i,1}, y_t^{i,2}, y_t^{i,3})$  such that

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

- We then formulate the (log)likelihood function

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

- 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  $p_t^i(\bar{n})$ ,  $p_t^i(\underline{n})$  and  $p_t^i(0)$  until convergence.

## Training errors and running time

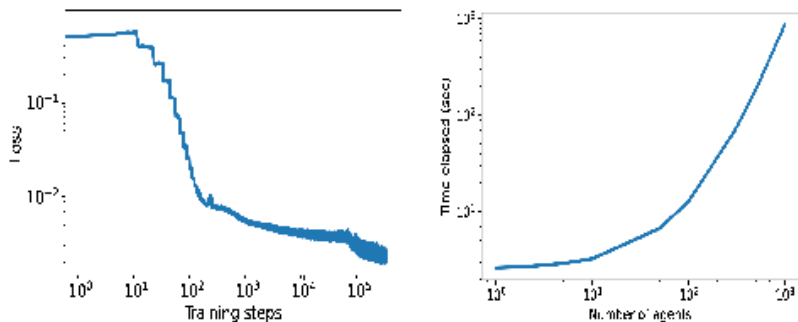


Figure 10. Training errors and running time for three-state employment model.

# The solution for the divisible labor model

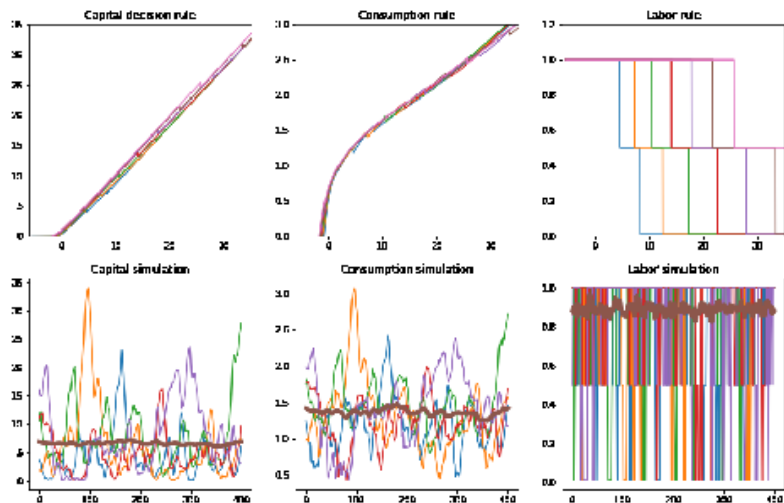


Figure 11. Solution to the three-state employment model.

# Conclusion

- We show 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.
- Here we've investigated 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.
- We are working on incorporating the deep learning classification approach into an overlapping generations model with retirement and employment decisions.

Thank you!

## 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, Maliar and Winant (JME 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)

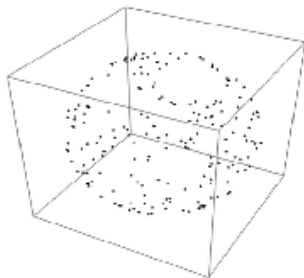


Figure 1. Hypercube versus hypersphere.

- 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:

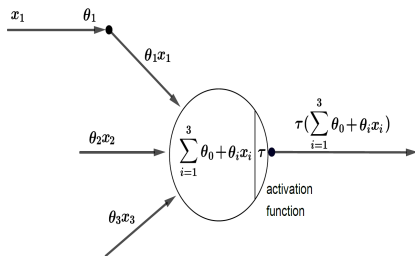


Figure 2a. Artificial neuron.

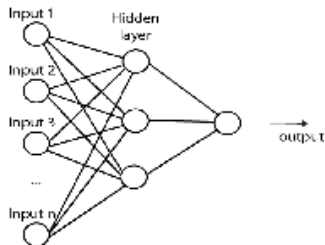


Figure 2b. Neural network.

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  $\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\theta_0+\theta_1x_1+\theta_2x_2+\dots+\theta_nx_n}}$ .

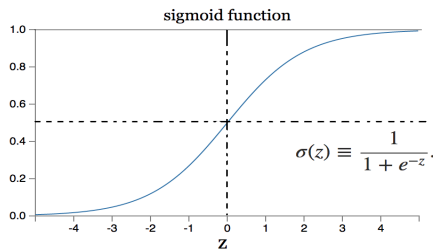


Figure 3. Sigmoid function.

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

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

where  $\varphi(\cdot)$  is a multilayer neural network parameterized by a vector of coefficients  $\theta$  (weights and biases),  $\sigma(z) = \frac{1}{1+e^{-z}}$  is a sigmoid function and  $\zeta_0$  is a constant term.

- In addition, we parameterize the Lagrange multiplier  $\mu_t^i$  associated with the borrowing constraint using an exponential activation function

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

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  $\ell$  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:

$$\Xi(\theta) \equiv E_{(K_t, Y_t, z_t)} \left\{ \left[ \Psi^{FB} \left( 1 - \frac{c_t^i}{w_t^i}, 1 - \mu_t^i \right) \right]^2 + \varpi_n \left[ n_t^i - \left( L - \left[ \frac{(c_t^i)^{-\gamma} W_t v_t^i}{B} \right]^{-1/\eta} \right) \right]^2 + \varpi_\mu \left[ \frac{\beta E_{(\Sigma_{t+1}, \epsilon_{t+1})} \left[ (c_{t+1}^i)^{-\gamma} R_{t+1} \mid \Sigma_{t+1}, \epsilon_{t+1} \right]}{(c_t^i)^{-\gamma}} - \mu_t^i \right]^2 \right\},$$

where  $K \equiv (k^1, \dots, k^\ell)$  and  $Y \equiv (v^1, \dots, v^\ell)$  are state variables;  $z_t$  is aggregate productivity;  $\Sigma_{t+1} \equiv (\epsilon_{t+1}^1, \dots, \epsilon_{t+1}^\ell)$  the individual productivity shocks;  $\epsilon_{t+1}$  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  $\left[E_{(\Sigma_{t+1}, \epsilon_{t+1})} [\cdot]\right]^2$  nested inside another expectation  $E_{(K_t, Y_t, z_t)} [\cdot]$ .
- Constructing two nested expectation operators is costly because the inner expectation operator  $E_{(\Sigma_{t+1}, \epsilon_{t+1})} [\cdot]$  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

$$E_{(K_t, Y_t, z_t)} \left[ \left[ E_{(\Sigma_{t+1}, \epsilon_{t+1})} [\cdot] \right]^2 \right] \neq E_{(K_t, Y_t, z_t)} E_{(\Sigma_{t+1}, \epsilon_{t+1})} \left[ [\cdot]^2 \right].$$

- 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  $\left[E_{(\Sigma_{t+1}, \epsilon_{t+1})} [\cdot]\right]^2$  under one random draw  $(\Sigma_{t+1}, \epsilon_{t+1})$  with a product of two expectation functions  $\left[E_{(\Sigma'_{t+1}, \epsilon'_{t+1})} [\cdot]\right] \times \left[E_{(\Sigma''_{t+1}, \epsilon''_{t+1})} [\cdot]\right]$  under two uncorrelated random draws  $(\Sigma'_{t+1}, \epsilon'_{t+1})$  and  $(\Sigma''_{t+1}, \epsilon''_{t+1})$ .
- Since the two random draws are uncorrelated, the expectation operator can be taken outside of the  $\Xi$  expectation function.

## The objective function under AiO expectation operator

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

Thus, we are able to represent the studied model as an expectation function across a vector of random variables

$(K_t, Y_t, z_t, \Sigma'_{t+1}, \epsilon'_{t+1}, \Sigma''_{t+1}, \epsilon''_{t+1})$ ; 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  $\nabla_{\theta}\Xi(\theta) = \nabla_{\theta}E[\xi(\omega; \theta)] = E[\nabla_{\theta}\xi(\omega; \theta)]$ , where  $\nabla_{\theta}$  is a gradient operator.
- The latter expectation function can be approximated by a simple average across Monte Carlo random draws  $E[\nabla_{\theta}\xi(\omega; \theta)] \approx \frac{1}{N} \sum_{n=1}^N \nabla_{\theta}\xi(\omega_n; \theta)$ , where  $\omega_n$  denotes a specific realization of the vector of random variables.
- Thus, the gradient descent method can be implemented as

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

where  $\theta$  and  $\lambda$  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  $\varphi\left(k_t^i, v_t^i, \{k_t^i, v_t^i\}_{i=1}^\ell, z_t; \theta\right)$  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  $\left(\{k_0^i, v_0^i\}_{i=1}^{\ell}, z_0\right)$  and parameterize three decision functions by a neural network with three outputs

$$\left\{\frac{n_t^i}{L}, \frac{c_t^i}{w_t^i}\right\} = \sigma\left(\zeta_0 + \varphi\left(k_t^i, v_t^i, \{k_t^i, v_t^i\}_{i=1}^{\ell}, z_t; \theta\right)\right),$$
$$\mu_t^i = \exp\left(\zeta_0 + \varphi\left(k_t^i, v_t^i, \{k_t^i, v_t^i\}_{i=1}^{\ell}, z_t; \theta\right)\right),$$

where  $w_t^i \equiv R_t k_t^i + W_t v_t^i n_t^i$  is wealth;  $\mu_t^i$  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;  $\zeta_0$  is a constant;  $\theta$  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  $\left(k_t^i, v_t^i, \{k_t^i, v_t^i\}_{i=1}^\ell, z_t\right) \equiv s_t^i$ , compute  $n_t^i, \mu_t^i, \frac{c_t^i}{w_t^i}$  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  $i = 1, \dots, \ell$ .

Step 2: (Construction of Euler residuals).

Draw two random sets of individual productivity shocks  $\Sigma_1 = (\epsilon_1^1, \dots, \epsilon_1^\ell)$ ,  $\Sigma_2 = (\epsilon_2^1, \dots, \epsilon_2^\ell)$  and two aggregate shocks  $\epsilon_1, \epsilon_2$ , and construct Euler residuals

$$\Xi(\theta) = \left\{ \left[ \Psi^{FB} \left( 1 - \frac{c_t^i}{w_t^i}, 1 - \mu_t^i \right) \right]^2 + \varpi_n \left[ n_t^i - \left( L - \left[ \frac{(c_t^i)^{-\gamma} W_t v_t^i}{B} \right]^{-1/\eta} \right) \right]^2 + \varpi_\mu \left[ \frac{\beta \left[ (c_{t+1}^i)^{-\gamma} R_{t+1} \mid \Sigma'_{t+1}, \epsilon'_{t+1} \right]}{(c_t^i)^{-\gamma}} - \mu_t^i \right] \left[ \frac{\beta \left[ (c_{t+1}^i)^{-\gamma} R_{t+1} \mid \Sigma''_{t+1}, \epsilon''_{t+1} \right]}{(c_t^i)^{-\gamma}} - \mu_t^i \right] \right\},$$

where  $\varpi_n, \varpi_\mu$  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  $\theta$  to minimize the residual function  $\Xi(\theta)$  by using a stochastic gradient descent method  $\theta \leftarrow \theta - \lambda \nabla_{\theta} \Xi(\theta)$  with  $\nabla_{\theta} \Xi(\theta) \approx \frac{1}{N} \sum_{n=1}^N \nabla_{\theta} \xi(\omega_n; \theta)$ , where  $n = 1, \dots, N$  denotes batches.

Step 4: (Simulation).

Move to  $t + 1$  by using endogenous and exogenous variables of Step 3 under  $\Sigma_1 = (\epsilon_1^1, \dots, \epsilon_1^{\ell})$  and  $\epsilon_1$  as a next-period state  $\left( \{k_{t+1}^i, v_t^i\}_{i=1}^{\ell}, z_{t+1} \right)$ .

## Algorithm 2: Deep learning for indivisible labor model

Algorithm 2: Deep learning for the indivisible labor model.

Step 0: (Initialization).

Construct initial state  $\left( \{k_0^i, v_0^i\}_{i=1}^{\ell}, z_0 \right)$  and parameterize the decision functions by

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

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  $\left(k_t^i, v_t^i, \{k_t^i, v_t^i\}_{i=1}^{\ell}, z_t\right)$ , compute  $n_t^i = \bar{n}$  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  $W_t$ ; and find  $k_{t+1}^i$  from the budget constraint for all agents  $i = 1, \dots, \ell$ .

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

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

Define  $y_t^i = \begin{cases} 1 & \text{if } \hat{n}_t^i = \bar{n}, \\ 0 & \text{otherwise,} \end{cases}$  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  $\Sigma_1 = (\epsilon_1^1, \dots, \epsilon_1^\ell)$ ,  $\Sigma_2 = (\epsilon_2^1, \dots, \epsilon_2^\ell)$  and two aggregate shocks  $\epsilon_1, \epsilon_2$ , to construct

$$\Xi(\theta) = \left\{ \left[ \Psi^{FB} \left( 1 - \frac{c_t^i}{w_t^i}, 1 - \mu_t^i \right) \right]^2 + \varpi_n \left[ y_t^i \ln(p(s_t^i; \theta)) + (1 - y_t^i) \ln(1 - p(s_t^i; \theta)) \right]^2 + \varpi_\mu \left[ \frac{\beta [(c_{t+1}^i)^{-\gamma} R_{t+1} | \Sigma'_{t+1}, \epsilon'_{t+1}]}{(c_t^i)^{-\gamma}} - \mu_t^i \right] \left[ \frac{\beta [(c_{t+1}^i)^{-\gamma} R_{t+1} | \Sigma''_{t+1}, \epsilon''_{t+1}]}{(c_t^i)^{-\gamma}} - \mu_t^i \right] \right\},$$

where  $\Psi^{FB}(a, b) = a + b - \sqrt{a^2 + b^2}$  is a Fischer-Burmeister function; and  $\varpi_n, \varpi_\mu$  are given weights.

Step 3: (Training).

...

Step 4: (Simulation).

## 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  $\left(\{k_0^i, v_0^i\}_{i=1}^{\ell}, z_0\right)$  and parameterize the decision functions by  $\left\{\frac{p_t^i(\bar{n})}{\Sigma}, \frac{p_t^i(\underline{n})}{\Sigma}, \frac{p_t^i(0)}{\Sigma}, \frac{c_t^i}{w_t^i}\right\} = \sigma\left(\zeta_0 + \varphi\left(k_t^i, v_t^i, \{k_t^i, v_t^i\}_{i=1}^{\ell}, z_t; \theta\right)\right)$ , where  $p_t^i(\bar{n})$ ,  $p_t^i(\underline{n})$  and  $p_t^i(0)$  are the probabilities to be full- and part-time employed and unemployed, respectively; and  $\Sigma \equiv p_t^i(\bar{n}) + p_t^i(\underline{n}) + p_t^i(0)$  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  $\left(k_t^i, v_t^i, \{k_t^i, v_t^i\}_{i=1}^{\ell}, z_t\right)$ , set  $n_t^i = \bar{n}$ ,  $n_t^i = \underline{n}$  and  $n_t^i = 0$  depending on which probability  $p_t^i(\bar{n})$ ,  $p_t^i(\underline{n})$  and  $p_t^i(0)$  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  $i = 1, \dots, \ell$ . Reconstruct  $V^E$ ,  $V^{PT}$  and  $V^U$ , respectively.

Find  $\hat{n}_t^i = \begin{cases} \bar{n} & \text{if } V^E = \max\{V^E, V^{PT}, V^U\}, \\ \underline{n} & \text{if } V^{PT} = \max\{V^E, V^{PT}, V^U\}, \\ 0 & \text{otherwise.} \end{cases}$

and define  $y_t^i = \begin{cases} (1, 0, 0) & \text{if } \hat{n}_t^i = \bar{n}, \\ (0, 1, 0) & \text{if } \hat{n}_t^i = \underline{n}, \\ (0, 0, 1) & \text{otherwise.} \end{cases}$  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  $\hat{n}_t^i = \begin{cases} \bar{n} & \text{if } V^E = \max \{V^E, V^{PT}, V^U\} \\ \underline{n} & \text{if } V^{PT} = \max \{V^E, V^{PT}, V^U\} \\ 0 & \text{otherwise.} \end{cases}$

Option 2: From discretized FOC  $\hat{n}_t^i = \begin{cases} \bar{n} & \text{if } L - \left[ \frac{c_i^{-\gamma} W_t \exp(v_t^i)}{B} \right]^{-1/\eta} \geq \bar{n}_f \\ \underline{n} & \text{if } L - \left[ \frac{c_i^{-\gamma} W_t \exp(v_t^i)}{B} \right]^{-1/\eta} \in [\bar{n}_p, \bar{n}_f] \\ 0 & \text{otherwise} \end{cases}$

Define  $y_t^i = \begin{cases} (1, 0, 0) & \text{if } \hat{n}_t^i = \bar{n}, \\ (0, 1, 0) & \text{if } \hat{n}_t^i = \underline{n}, \\ (0, 0, 1) & \text{otherwise.} \end{cases}$  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  $\Sigma_1 = (\epsilon_1^1, \dots, \epsilon_1^\ell)$ ,  $\Sigma_2 = (\epsilon_2^1, \dots, \epsilon_2^\ell)$  and two aggregate shocks  $\epsilon_1, \epsilon_2$ , and construct the residuals

$$\begin{aligned} \Xi(\theta) = & \left\{ \left[ \Psi^{FB} \left( 1 - \frac{c_t^i}{w_t^i}, 1 - \mu_t^i \right) \right]^2 \right. \\ & + \frac{\varpi_n}{3} \sum_{k=1}^3 \left[ \widehat{y}_t^{i,k} \ln \left( p \left( s_t^i; \theta^{(k)} \right) \right) + \left( 1 - \widehat{y}_t^{i,k} \right) \ln \left( 1 - p \left( s_t^i; \theta^{(k)} \right) \right) \right]^2 \\ & \left. + \varpi_\mu \left[ \frac{\beta \left[ (c_{t+1}^i)^{-\gamma} R_{t+1} \mid \Sigma'_{t+1}, \epsilon'_{t+1} \right]}{(c_t^i)^{-\gamma}} - \mu_t^i \right] \left[ \frac{\beta \left[ (c_{t+1}^i)^{-\gamma} R_{t+1} \mid \Sigma''_{t+1}, \epsilon''_{t+1} \right]}{(c_t^i)^{-\gamma}} - \mu_t^i \right] \right\}, \end{aligned}$$

where  $\Psi^{FB}(a, b) = a + b - \sqrt{a^2 + b^2}$  is a Fischer-Burmeister function; and  $\varpi_n, \varpi_\mu$  are given weights.

Step 3: (Training).

...

Step 4: (Simulation).