# Solving and Estimating Finite-time Dynamic Discrete Choice Models With Deep Learning

Matheus Silva

**ARTICLE HISTORY**
Compiled November 15, 2022

## 1. Introduction

**Note: This is preliminary work submitted as a writing sample.**

Finite-time discrete choice models are widespread in Economics. Classical examples include job search problems (Wolpin, 1987), patent renewal (Pakes, 1986), and the bus engine replacement (Rust, 1987). Despite the intuitive mathematical formulation of such models, closed-form solutions are black swans because of their nonstationary nature and numerical solutions may be prohibitively expensive to calculate due to the curse of dimensionality.

Hence, empirical researchers eventually face the crucial choice of how rich their models are, in terms of features. Often times this means ignoring relevant dynamics present in the data for the sake of obtaining a solution in human time. For example, Keane and Wolpin (1994) study a occupational choice model in which agents yearly choose between going to school, working at a firm, and working at home. Each of those alternatives yields stochastic payoffs that are possibly correlated over time in real life (for example, salaries likely exhibit a trend over time instead of being random). However, for the sake of computational tractability, Keane and Wolpin (1994) restrict such structural shocks to be independent and identically distributed. Economically, this means that salaries are not persistent over time.

So, say a researcher is studying labor supply and fertility decisions of females, it is reasonable to say that an unexpected pregnancy affects future decisions regarding education and work, being at home or not. Ignoring such dynamics may impact the conclusions of one's study significantly. Even more severe is the invalidity of the counterfactuals generated by the mathematical model used in the study. In this case, the policy suggestions based on these models may do more harm than good.

To the best of my knowledge, the state-of-art off-the-shelf software available to solve finite-time dynamic discrete choice models is *respy* (Eisenhauer, 2019). Despite its fast implementation, *respy* is limited to problems in the so-called EKW models

(Aguirregabiria and Mira, 2010).

In this paper, I propose a generalized approach to solving finite-time discrete choice models that uses neural networks to approximate their solution (a policy function). This exploits the fact that deep learning circumvents the curse of dimensionality (Berner et al. (2021) and Fernandez-Villaverde et al. (2021)), allowing us to quickly obtain an approximate solution to FTDCM with potentially large state spaces. Considering a model within the EKG framework, the deep learning solution requires less computing time compared to *respy*. I then relax the i.i.d. assumption in Keane and Wolpin (1994) and repeat their estimation exercise using this new method.

This paper fits into an expanding literature that uses machine learning methods to solve economic models. Recent examples include works by Maliar and Maliar (2020) which focuses on approximating a solution to a general equilibrium model à la Krussel and Smith (1998?). Fernandez-Villaverde et al. (2021) focuses on solving an extension of Lucas and Prescott's (1971) general equilibrium model. Like much of the machine learning literature, many of the papers in this area employ methods to solve what is the equivalent of a fixed-point problem. These problems are not of interest in this paper because the solution of dynamic choice discrete models are not stationary as they depend on how close the agent is to the last period they make an economic decision. A few papers deal with nonstationary solutions to similar problems in other contexts, but their methodologies relate closely to this work. For example, Han and E (2016) focus on distribution of electricity between a generator and its potential clients and Achdou et al (2021) approximate the solutions of parabolic differential equations that solve a continuous-time economic model.

In what follows, I describe the general setup of a FTDCM as in Keane and Wolpin (1994), present and discuss the architecture of the neural network in question. I proceed to discussing why approximating policy functions with deep learning avoids dealing with the curse of dimensionality. Finally, I solve Keane and Wolpin's (1994) model and repeat their estimation exercise.

## 2. The Finite-time Discrete Choice Problem

Let $\mathcal{T} \equiv \{0, 1, 2, \ldots, T-1\}$ be a finite set of $T$ time periods. An economic agent may choose one action $k$ among $\mathcal{K}$ possible alternatives at each time period $t \in \mathcal{T}$. Let $d_k(t)$, $\forall t \in \mathcal{T}$, be an indicator function,

$$d_k(t) \equiv \begin{cases} 1, & \text{if the agent chooses } k \text{ at time } t \\ 0, & \text{otherwise} \end{cases}$$

Each agent observes a set of states at time $t$, denoted $\Omega(t)$. These states include all the variables that influence their decisions. In Keane and Wolpin (1994), those are

the structural shocks that determine the utility values of working at home, working outside of the home, and working at home. Let the structural shocks at time $t$ be denoted $\xi(t)$ and the instantaneous utility of the $k-th$ alternative is denoted $R_k(t)$. It is worthwhile to point out that the problem of estimation is separate from the issue of solving the model since the econometrician cannot observe the whole vector of states. I will be study the estimation problem in depth in **Section XX**.

With all those definitions in place, one can write the intertemporal utility maximization problem the agent solves as:

$$\max \mathbb{E}\left[\sum_{\tau=t}^{T} \beta^{\tau-t} \sum_{k\in K} R_k(t) d_k(t) | \Omega(t)\right] \tag{1}$$

$$s.t. \begin{cases} \sum_{k\in\mathcal{K}} d_k(t) = 1 \ \forall t & (a) \\ \Omega(t+1) = \phi(\Omega(t), d_k(t), \xi(t)) & (b) \\ \Omega(0) \text{ given} \end{cases} \tag{2}$$

By choosing $d_k(t)$ given states $\Omega(t)$ and the instantaneous rewards $R_k(t)$ in all $t \in \mathcal{T}$. Given $\Omega(0)$, the agents are subject to two restrictions. Restriction $(a)$ implies only one alternative within the choice set can be picked at each time. Restriction $(b)$ is a law of motion that describes how the states $\Omega(t)$ evolve over time. In this general formulation, the state vector evolves according to some function $\phi$ that depends on the current state $\Omega(t)$, the action taken by the individual at that specific time period $d_k(t)$, and a random shock $\xi(t)$.

## 2.1. *Examples*

It is illustrative to see the general model applied in different contexts. I will show three examples I brought up previously.

### *2.1.1. Wolpin (1987)*

Finite-time job search model.

Setup:

- Initial state:

$$\Omega_0 = \begin{pmatrix} \xi_0 \\ 1 \end{pmatrix}$$

- State transition law:

$$\Omega_{t+1} = \begin{cases} \begin{pmatrix} \xi_{t+1} \\ 1 \end{pmatrix} & \text{if } d_t = 1 \text{ (rejected offer)} \\ \begin{pmatrix} \xi_t \\ 0 \end{pmatrix} & \text{if } d_t = 0 \text{ (accepted offer)} \end{cases}$$

- Instant intermediary cost function:

$$c_t = \beta^{t-1}[d_t * \xi_t - c * d_{t-1}]$$

- Layers of the neural network:
  1. Linear
  2. Softmax (this is the approximation for the argmax!)

### *2.1.2. Keane and Wolpin (1994)*

Within the setup in (1) with a slight change of notation...:

Rewards:

$$R_1(t) = \left[\alpha_{1,0} + \alpha_{1,1}s_t + \alpha_{1,2}x_{1t} + \alpha_{1,3}x_{1t}^2 + \alpha_{1,4}x_{2t} + \alpha_{1,5}x_{2t}^2\right] exp(\varepsilon_{1t})$$
$$R_2(t) = \left[\alpha_{2,0} + \alpha_{2,1}s_t + \alpha_{2,2}x_{1t} + \alpha_{2,3}x_{1t}^2 + \alpha_{2,4}x_{2t} + \alpha_{2,5}x_{2t}^2\right] exp(\varepsilon_{1t})$$
$$R_3(t) = \alpha_{3,0} - \alpha_{3,6}I(s_t \geq 12) - \alpha_{3,7}d_3(t-1) + \varepsilon_{3t}$$
$$R_4(t) = \alpha_{4,0} + \varepsilon_{4t}4$$

State:

$$\Omega(t) = \begin{bmatrix} x_{1t} \\ x_{2t} \\ s_t \\ d_3(t-1) \\ 1 \\ I(s_t \geq 12) \end{bmatrix}$$

Transition:

$$S(t+1) = S(t) + \begin{bmatrix} d_1(t) \\ d_2(t) \\ d_3(t) \\ d_3(t) - S_t^3 \\ 0 \\ I(d_3(t) + S_t^3 \geq 12) \end{bmatrix}$$

Write rewards in matrix quadratic form:

$$R_1(t) = S_t' exp(\varepsilon_{1t}) \begin{pmatrix} \alpha_{1,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha_{1,5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} S_t + exp(\varepsilon_{1t}) \begin{pmatrix} \alpha_{1,2} & \alpha_{1,4} & \alpha_{1,1} & 0 & \alpha_{1,0} & 0 \end{pmatrix} S_t$$

$$R_1(t) = S_t' A_1(t) S_t + B_1(t) S_t$$

$$R_2(t) = S_t' exp(\varepsilon_{2t}) \begin{pmatrix} \alpha_{2,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha_{2,5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} S_t + exp(\varepsilon_{2t}) \begin{pmatrix} \alpha_{2,2} & \alpha_{2,4} & \alpha_{2,1} & 0 & \alpha_{2,0} & 0 \end{pmatrix} S_t + \varepsilon_{3t}$$

$$R_2(t) = S_t' A_2(t) S_t + B_2(t) S_t + \varepsilon_{3t}$$

$$R_3(t) = S_t' \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} S_t + \begin{pmatrix} 0 & 0 & 0 & -\alpha_{3,7} & \alpha_{3,0} & -\alpha_{3,6} \end{pmatrix} S_t + \varepsilon_{4t}$$

$$R_3(t) = S_t' A_3 S_t + B_4 S_t + \varepsilon_{4t}$$

$$R_4(t) = S_t' \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} S_t + \begin{pmatrix} 0 & 0 & 0 & 0 & \alpha_{4,0} & 0 \end{pmatrix} S_t$$

$$R_4(t) = S_t' A_4 S_t + B_4(t) S_t + \varepsilon_{4t}$$

## 3. The Neural Network Architecture

Due to being a finite-horizon problem, the policy functions are time specific. Solving this type of mathematical problem with a neural network is a departure from the current literature on this broad topic, which deals almost exclusively with infinite-

horizon problems. In this latter case, solving the model consists in finding the best hyper-parameters of a single recurrent neural network. Finding this stable RNN is equivalent to finding a fixed-point in the space of functions.

To describe the neural network, it is convenient to detail the time structure of the model. At $t = 0$, the agent observes their initial state $S(0)$ and makes the decision $d_k(0)$, after which the period ends; at $0 < t < T - 1$, the agent observes $S(t)$ (which depends on $S(t-1)$, the agent's decision $d_k(t-1)$, and a stochastic shock, $\xi(t)$,) and makes the decision $d_k(t)$. This procedures until the last period, when the agent faces a boundary condition. The goal is to approximate the policy function $d_k(t)$ for every $t$ using a neural network that I denote $H(\alpha_t)$. Notice that $H$ is a fixed strucure that depends on time-varying hyper-parameters $\alpha_t$. This allows us to deal with the nonstationary nature of the problem. Moreover, let $h^n(\alpha_t)$ be the $n-th$ layer (out of a total of $N$ layers) of $H(\alpha_t)$.

Graphically, this neural network has the following architecture:

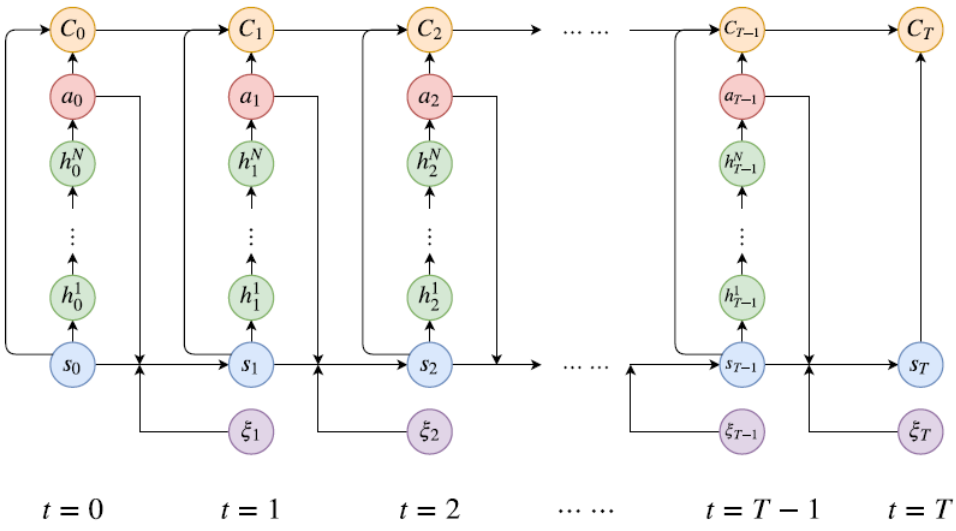


**Figure 1.** Network architecture from Han and E (2016)

The set of hyper-parameters is then obtained by backpropagation using the gradient descent algorithm.

### 3.1. *A simple, cake-eating problem*

This neural network can also be used to solve problems where the policy function is not discrete (by using, say, a linear activation layer instead of a softmax one.) Here, I want to illustrate the solution of the following cake-eating problem,

$$V_0(s_0) = \max_{\{c_t\}_{t=0}^{T}} \mathbb{E}\left[\sum_{t=0}^{T} \beta^t ln(c_t)\right]$$

$$s.t. \begin{cases} \xi_\tau = \begin{cases} 0.1, prob = p \\ 0, prob = 1-p \end{cases} & (0 < \tau \leq T) \\ c_t \geq 0 \\ s_{t+1} = s_t - c_t + \xi_{t+1} \\ given\ s_0 \geq 0 \end{cases}$$

*Implementation details*

We want to approximate the true value function $c_t(s)$ with a neural network $c_t(s_t|\theta_t)$, where $\theta_t$ are its hyper-parameters at time $t$. Here, I use a simple, linear NN: $c_t(s_t|\theta_t) = \theta_t s_t$.

The intermediary cost function is:

$$C_t \equiv \sum_{\tau=0}^{t} \beta^\tau ln(c_\tau)$$

Define the "future" intermediary cost function (for the backpropagation step.)

$$C^t \equiv \sum_{\tau=t}^{T} \beta^\tau ln(c_\tau)$$

Backpropagation for some parameter $\theta_\tau$[1]:

$$\nabla_\tau = \frac{\partial C^t}{\partial \theta_\tau} = \sum_{\tau=t}^{T} \beta^\tau \frac{W_\tau s_t}{s_{\tau+1}}$$

$$W_t \equiv 1$$

$$W_\tau \equiv \prod_{i=\tau+1} W_{\tau-1}(1-\theta_\tau)$$

[1]Obtain this by taking the total derivative of $C^t$ with respect to the parameter of interest. You MUST take into consideration the state transition law!

| $T$ | $\alpha_1$ | $\alpha_2$ | $\alpha_3$ | $\alpha_4$ |
|---|---|---|---|---|
| 2 | 0.5 | - | - | - |
| 3 | 0.3333 | 0.5 | - | - |
| 4 | 0.25 | 0.3333 | 0.5 | - |
| 5 | 0.2 | 0.25 | 0.3333 | 0.5 |

**Table 1.** Deep learning solution without uncertainty ($p = 0$, $50,000$ samples)

| $T$ | $\alpha_1$ | $\alpha_2$ | $\alpha_3$ | $\alpha_4$ |
|---|---|---|---|---|
| 2 | 0.5021 | - | - | - |
| 3 | 0.3351 | 0.5028 | - | - |
| 4 | 0.2515 | 0.3356 | 0.5037 | - |
| 5 | 0.2012 | 0.2517 | 0.3360 | 0.5042 |

**Table 2.** Deep learning solution with uncertainty ($p = 0.05$, $50,000$ samples)

These solutions coincide with the analytic ones. See appendix for Python code. **Discussion:** In this case, the method works really well because the true solution is, indeed, a linear function.