The Econometrics Journal

Econometrics Journal (2020), volume **23**, pp. S1–S24.
doi: 10.1093/ectj/utaa005

Artificial intelligence as structural estimation: Deep Blue, Bonanza, and AlphaGo

MITSURU IGAMI

Yale Department of Economics, 37 Hillhouse Avenue, New Haven, CT 06511, United States of America.
Email: mitsuru.igami@yale.edu

First version received: 13 October 2018; final version accepted: 29 January 2020.

Summary: This article clarifies the connections between certain algorithms to develop artificial intelligence (AI) and the econometrics of dynamic structural models, with concrete examples of three ‘game AIs’. Chess-playing Deep Blue is a calibrated value function, whereas shogi-playing Bonanza is an estimated value function via Rust’s nested fixed-point (NFXP) method. AlphaGo’s ‘supervised-learning policy network’ is a deep-neural-network implementation of the conditional-choice-probability (CCP) estimation reminiscent of Hotz and Miller’s first step; the construction of its ‘reinforcement-learning value network’ is analogous to their conditional choice simulation (CCS). I then explain the similarities and differences between AI-related methods and structural estimation more generally, and suggest areas of potential cross-fertilization.

Keywords: *Approximate dynamic programming*, *artificial intelligence*, *conditional choice probability*, *deep neural network*, *dynamic structural model*, *inverse reinforcement learning*, *optimal control*, *reinforcement learning*, *simulation estimator*.

1. INTRODUCTION

Most economists have a high-level understanding of artificial intelligence (AI) and reinforcement learning (RL).[1] Some are even aware of their similarities to structural estimation (SE) in econometrics, but only vaguely. A good command of details is necessary to assess AI/RL’s usefulness for economic research and to find opportunities for cross-fertilization. This article clarifies the connections between AI/RL and the estimation of dynamic structural models with

The first version of this paper was circulated on October 30, 2017, under a slightly longer title. I thank Susan Athey, John Rust, Simon Scheidegger, and Takuo Sugaya for many detailed suggestions. This paper also benefitted from seminar comments at Riken AIP, Georgetown University, Tokyo University, Osaka University, Harvard University, MIT, Johns Hopkins University, Stanford GSB, and ‘The Third Cambridge Area Economics and Computation Day Conference’ at Microsoft Research New England, ‘The Second Conference on Dynamic Structural Models: Methodologies and Applications of Structural Dynamic Models and Machine Learning’ at University of Copenhagen, and 2018 ‘NBER Economics of Artificial Intelligence Conference’ at University of Toronto, as well as conversations with Xiaohong Chen, Victor Chernozhukov, Philip Haile, Jerry Hausman, Guido Imbens, Greg Lewis, Robert Miller, Yusuke Narita, Aviv Nevo, Whitney Newey, Anton Popov, Jeff Qiu, Elie Tamer, Kosuke Uetake, Hal Varian, and Yosuke Yasuda.

[1] The formal definition of AI seems contentious, partly because scholars have not agreed on the definition of intelligence in the first place. This paper follows a broad definition of AI as computer systems able to perform tasks that traditionally required human intelligence. Likewise, I use the term ‘machine learning’ and related jargons without defining them *whenever the original developers of the programs used them* to describe their procedures.

concrete examples. Specifically, I explain the algorithms and development procedures for three famous ‘game AIs’ for the two-player board games of chess, shogi (Japanese chess), and Go. These games represent a well-defined environment that is conceptually clean but computationally challenging, which is an ideal setting for comparing and contrasting AI/RL and SE.[2]

In chess, IBM’s computer system named Deep Blue defeated Grandmaster Garry Kasparov in 1997. In shogi, a machine-learning-based program called Bonanza challenged (and was defeated by) Ryūō champion Akira Watanabe in 2007, but one of its successors (Ponanza) played against Meijin champion Amahiko Satoh and won in 2017. In Go, DeepMind developed AlphaGo, a deep-reinforcement-learning-based program, which beat the 2-dan European champion Fan Hui in 2015, a 9-dan (highest rank) professional Lee Sedol in 2016, and the world’s best player, Ke Jie, in 2017. Its successors, AlphaGo Zero and AlphaZero, are even stronger.

The development procedures of these game AIs are closely related to the estimation methods for dynamic structural models, such as Rust (1987), Hotz and Miller (1993), and Hotz et al. (1994). Their similarities stem from the fact that both literatures build on the mathematical framework of dynamic optimization. A typical AI/RL project tries to find an approximately optimal solution to a given dynamic programming (DP) problem; a typical SE project tries to infer agents’ objective functions by assuming that their observed behaviours in the data are approximately optimal. Hence, the two literatures work on the inverse problems of each other’s.

This overlap suggests potential ‘gains from trade’. First, many AI/RL methods use functional approximation and/or stochastic approximation to handle high-dimensional state spaces. Neither is new to econometrics. Nonparametric regressions and Monte Carlo simulations have been used for decades. Nevertheless, the recent success of game AIs in achieving super-human performance suggests a broader applicability of these techniques for high-dimensional problems of practical importance.

Second, the guiding principles of SE could potentially discipline the hitherto ad hoc attempts to ‘explain’ AI in the non-economics literature. The purpose of a typical research in SE is threefold: (i) to understand the agents’ motives behind their behaviours, (ii) to understand the mechanisms through which outcomes are generated by various economic forces, and (iii) to evaluate alternative ‘rules of the game’ (e.g., public policy) to improve aggregate outcomes (e.g., social welfare). The components of aggregate performance in (iii) are identified and estimated in step (i), and their interactions are analyzed and explained in step (ii). These steps of SE embody a framework through which economists *explain* complicated real-world phenomena. Hence, SE provides a good starting point to think about what (truly) ‘explainable AI’ should look like.

I organize the rest of the paper as follows. After introducing basic notations in Section 2, I describe the main components of Deep Blue, Bonanza, and AlphaGo in Sections 3, 4, and 5, respectively, and explain their counterparts in the SE literature. Section 6 describes RL (and approximate dynamic programming) more generally. Section 7 explains SE and discusses its similarities with and differences from RL. Section 8 concludes.

In the literature, the most closely related papers are those that proposed the econometric methods for estimating dynamic structural models, such as Rust (1987), Hotz and Miller (1993), and Hotz et al. (1994). The game AIs that I analyse in this paper are probably the most successful (or the most popular) empirical applications of these methods. For a historical review of numerical methods for dynamic programming, see Rust (2017, 2019).[3] At a higher level, the purpose of this

[2] Simpler examples from one-player video games are too trivial for economic modelling and estimation. More complex real-world applications involve too many additional issues and would blur the focus.

[3] Norets (2012) proposes an estimation approach for dynamic discrete-choice models by combining the Bayesian Markov chain Monte Carlo and artificial neural networks.

paper is to clarify the connections between machine learning and econometrics in certain areas. Hence, the paper is similar in spirit to, for example, Belloni et al. (2014), Varian (2014), Athey (2017), and Mullainathan and Spiess (2017), among many others in the growing literature at the intersection of computer science and economics. This paper has a unique focus on SE, DP, and RL.[4]

2. MODEL

Some of the AI-related methods are branded as 'model-free', but they are not. All game AIs must operate within the environments that are specified by the rules of the games, and their development procedures exploit the nature of these games as well as the mathematical properties of DP.

Chess, shogi, and Go belong to the same class of games, with two players ($i = 1, 2$), discrete time ($t = 1, 2, ...$), alternating moves (players 1 and 2 choose their actions, a_t, in odd and even periods, respectively), perfect information, and deterministic state transition,

$$s_{t+1} = f\left(s_t, a_t\right), \tag{2.1}$$

where both the transition, $f(\cdot)$, and the initial state, s_1, are completely determined by the rules of each game.[5]

Action space is finite and is defined by the rule as 'legal moves',

$$a_t \in \mathcal{A}\left(s_t\right). \tag{2.2}$$

State space is finite as well, and consists of four mutually exclusive subsets:

$$s_t \in \mathcal{S} = \mathcal{S}_{cont} \sqcup \mathcal{S}_{win} \sqcup \mathcal{S}_{loss} \sqcup \mathcal{S}_{draw}, \tag{2.3}$$

where I denote 'win' and 'loss' from the perspective of player 1 (e.g., player 1 wins and player 2 loses if $s_t \in \mathcal{S}_{win}$). Neither player wins if $s_t \in \mathcal{S}_{draw}$. The game continues as long as $s_t \in \mathcal{S}_{cont}$.

The two players' payoffs sum to zero:

$$u_1\left(s_t\right) = \begin{cases} 1 & \text{if } s_t \in \mathcal{S}_{win}, \\ -1 & \text{if } s_t \in \mathcal{S}_{loss}, \text{ and} \\ 0 & \text{otherwise,} \end{cases} \tag{2.4}$$

with $u_2(s_t)$ defined in a similar manner (but with win/loss payoffs flipped). This setup means that chess, shogi, and Go are well-defined finite games. In principle, such games can be solved exactly and completely by backward induction from the terminal states.[6]

In practice, even today's high-performance computers cannot solve them within our lifetime, because the size of the state space, $|\mathcal{S}|$, is large.[7] The approximate values of $|\mathcal{S}|$ for chess, shogi,

[4] In the context of RL and optimal control, Watkins (1989) clarified the connection between RL and DP in his PhD dissertation that proposed Q-learning. See Bertsekas and Tsitsiklis (1996), Sutton and Barto (1998, 2018), Powell (2011), and Bertsekas (2019) for general reference.

[5] This setup abstracts from the time constraints in official games, because the developers of game AIs typically do not incorporate them at the data-analysis stage. Hence, t represents turn-to-move, not clock time. Appendix C investigates this issue.

[6] According to Schwalbe and Walker (2001), Zermelo's (1913) theorem says that in finite two-player games with alternating moves, zero-sum payoffs, and perfect information, either player 1 can force a win, player 2 can force a win, or both players can force at least a draw. Whether a unique subgame-perfect equilibrium exists is unknown, because whether multiple winning strategies exist has not been proved.

[7] See https://www.top500.org/ for the current frontier of high-performance computing.

and Go are 10^{47}, 10^{71}, and 10^{171}, respectively, which are comparable to the number of atoms in the observable universe ($10^{78} \sim 10^{82}$) and certainly larger than the total information-storage capacity of humanity (of the order of 10^{21} bytes).[8]

3. CHESS: DEEP BLUE

3.1. Algorithms and development procedures

IBM's Deep Blue is a computer system with custom-built hardware and software components. I focus on the latter, programming-related part. Deep Blue's program consists of three key elements: an evaluation function, a search algorithm, and databases.

Evaluation function. The 'evaluation function' of Deep Blue is a linear combination of certain features of the current board state s_t. It quantifies the probability of eventual winning ($\Pr_{win}$) or its monotonic transformation, $g\left(\Pr_{win}\right)$:

$$V_{DB}\left(s_t;\theta\right) = \theta_1 x_{1,t} + \theta_2 x_{2,t} + \cdots + \theta_K x_{K,t}, \tag{3.1}$$

where $\theta \equiv (\theta_1, \theta_2, \ldots, \theta_K)$ is a vector of K parameters, and $x_t \equiv (x_{1,t}, x_{2,t}, \ldots, x_{K,t})$ is a vector of K observable characteristics of s_t. The published version featured $K = 8{,}150$ parameters (Campbell, Hoane and Hsu, 2002).

A typical evaluation function for computer chess considers the 'material value' associated with each type of piece, such as 1 point for a pawn, 3 points for a knight, 3 points for a bishop, 5 points for a rook, 9 points for a queen, and arbitrarily many points for a king (e.g., 200 or 1 billion), because the game ends when a king is captured. Other factors include the relative positions of these pieces, such as pawn structure, protection of kings, and the opinion of experts that a pair of bishops is usually worth more than the sum of their individual material values. Finally, the importance of these factors may change depending on the phase of the game: opening, middle, or endgame.

Reasonable parameterization and the choice of board characteristics (variables) require expert knowledge. Multiple Grandmasters (the highest rank of chess players) advised the Deep Blue development team. However, they did not use statistical analysis or data from professional players' games. In other words, they did not estimate θ in the econometric sense. Each of the 8,150 parameters, θ, was *manually adjusted* until the program's performance reached a satisfactory level.

Search algorithm. The second component of Deep Blue is 'search', or a solution algorithm to choose the optimal action at each turn to move. In its 'full-width-search' procedure, the program evaluates every possible position for a fixed number of future moves along the game tree, using the 'minimax algorithm' and some 'pruning' methods. This 'search' is a version of numerical backward induction.

Databases. Deep Blue uses two databases: one for the endgame and the other for the opening phase.

The endgame database embodies the cumulative efforts by the computer chess community to solve the game in an exact manner. Ken Thompson and Lewis Stiller developed Deep

[8] In 2018, the world's hard disk drive (HDD) industry shipped less than a zetabyte, or 10^{21} bytes, in total.

Blue's endgame database with all five-piece positions (i.e., the states with only five pieces, and all possible future states that can be reached from them), as well as selected six-piece positions.[9]

The second database is an 'opening book', which is a collection of common openings (i.e., move patterns at the beginning of the game) that experts consider good plays. It also contains good ways to counter the opponent's poor openings, again based on the judgment of experts. Grandmasters Joel Benjamin, Nick De Firmian, John Fedorowicz, and Miguel Illescas created one with about 4,000 positions, by hand.

The team also used some data analysis to prepare an 'extended book' to guard against non-standard opening positions. The description in Campbell et al. (2002) suggests that they constructed a parametric move-selection function based on the data analysis of 700,000 human games. They even incorporated annotators' commentary on the moves. Nevertheless, the use of data analysis seems limited to this back-up database.

Performance. Deep Blue lost a match to the top-ranked Grandmaster Garry Kasparov in 1996, but defeated him in 1997. Since then, the use of computer programs has become widespread in terms of both training and games (e.g., those played by hybrid teams of humans and computers).[10]

3.2. Deep Blue is a calibrated value function

The fact that IBM manually adjusted the parameter θ by trial and error means that Deep Blue is the fruit of painstaking efforts to 'calibrate' a value function with thousands of parameters: Deep Blue is a calibrated value function.

A truly optimal value function would obviate the need for any forward-looking and backward-induction procedures to solve the game (i.e., the 'search algorithm'), because the optimal value function should embody such a solution. However, the parametric value function is merely an attempt to approximate the optimal one. Approximation errors (or misspecification) leave room for performance improvement by studying the current (exact) state with the additional use of backward induction.

The 'full-width search' procedure is a brute-force numerical search for the optimal choice by backward induction, but solving the entire game is infeasible. Hence, this backward induction is performed on a *truncated* version of the game tree (truncated at some length L from the current turn t). The 'terminal' values at turn $(t + L)$ are given by the parametric value function (3.1). The opponent is assumed to share the same terminal-value function at $(t + L)$ and to choose its move to minimize the focal player's $V_{DB}(s_{t+L}; \theta)$.

Thus, Deep Blue is a parametric (linear) function to approximate the winning probability at the end of a truncated game tree, $V_{DB}(s_{t+L};\ \theta)$, in which the opponent shares exactly the same value function and the time horizon. In other words, Deep Blue is a calibrated, approximate terminal-value function in a game that the program plays against its doppelgänger.

[9] A database with all five-piece endings requires 7.05 gigabytes of hard disk space; storing all six-piece endings requires 1.2 terabytes.

[10] See Kasparov (2007), for example.

4. SHOGI: BONANZA

4.1. Algorithms and development procedures

In 2005, Kunihito Hoki, a chemist who specialized in computer simulations at the University of Toronto, spent his spare time developing a shogi-playing program named Bonanza, which won the world championship in computer shogi in 2006. Hoki's Bonanza revolutionized the field of computer shogi by introducing machine learning to 'train' (i.e., estimate) a more flexible evaluation function than those used for chess and those designed for the existing shogi programs.[11]

More complicated evaluation function. The developers at IBM could manually adjust 8,150 parameters in $V_{DB}(s_t; \theta)$ and beat the human chess champions. The same approach did not work for shogi. Shogi programs before Bonanza could only compete at amateur level at best. This performance gap between chess and shogi AIs is rooted in the more complicated state space of shogi, with $\left|\mathcal{S}_{shogi}\right| \approx 10^{71} > 10^{47} \approx |\mathcal{S}_{chess}|$.

Several factors contribute to the complexity of shogi: a larger board size ($9 \times 9 > 8 \times 8$), more pieces ($40 > 32$), and more types of pieces ($8 > 6$). In addition, most of the pieces have limited mobility[12] and, other than kings, never die. Non-king pieces are simply 'captured', not killed, and can then be 'dropped' (redeployed on the capturer's side) almost anywhere on the board at any of the capturer's subsequent turns. This last feature is particularly troublesome for an attempt to solve the game exactly, because the effective $|\mathcal{S}|$ does not decrease over time.[13]

Hoki designed a flexible evaluation function by factorizing the positions of pieces into (a) the positions of any three pieces including the two kings, and (b) the positions of any three pieces including only one king. This granular characterization turned out to capture important features of the board through the relative positions of three-piece combinations. Bonanza's evaluation function, $V_{BO}(s_t; \theta)$, also incorporated other, more conventional characteristics, such as individual pieces' material values (see Hoki and Watanabe, 2007, pp. 119–120, for details). As a result, $V_{BO}(s_t; \theta)$ has a linear functional form similar to (3.1) but contains $K = 50$ *million* variables and the same number of parameters (Hoki, 2012).

Machine learning (logit-like regression). That the Deep Blue team managed to adjust thousands of parameters for the chess program by human hand is almost incredible. But the task becomes impossible with 50 million parameters. Hoki gave up on manually tuning Bonanza's θ and chose to use statistical methods to automatically adjust θ based on the data from the 50,000 games of professional shogi players on official record: supervised learning.

Each game takes 100 moves, on average. Hence, the data contain approximately 5 million pairs of (a_t, s_t).[14] The reader might notice that the sample size is smaller than $|\theta|$ (50 million). Hoki

[11] Hoki acknowledges that machine-learning methods had previously been used in computer programs to play backgammon and reversi ('Othello'), but says he could not find any successful applications to chess or shogi in the literature (Hoki and Watanabe, 2007).

[12] Four of the eight types of pieces in shogi can move only one unit distance at a time, whereas only two of the six types of pieces in chess (pawn and king) have such low mobility. The exact positions of pieces become more important in characterizing the state space when mobility is low, whereas high mobility makes pure 'material values' relatively more important, because pieces can be moved to wherever they are needed within a few turns.

[13] In other words, shogi is not a *directional dynamic game*. See Iskhakov et al. (2016) for an algorithm to solve such games.

[14] In earlier versions of Bonanza, Hoki also used additional data from 50,000 unofficial, online game records as well, to cover some rare states such as nyuu-gyoku positions (in which a king enters the opponent's territory and becomes

reduced the effective number of parameters by additional restrictions to 'stabilize the numerical optimization process' (i.e., restrictions on the parameter space).

Like Deep Blue, Bonanza chooses its move at each of its turns t by searching for the action a_t that maximizes V_{BO} in some future turn $t + L$,

$$a_t^* = \arg \max_{a \in \mathcal{A}(s_t)} \{V_{BO}(s_{t+L}; \theta)\}, \tag{4.1}$$

assuming that the opponent shares the same terminal-value function and tries to minimize it. Note that the 'optimal' choice a_t^* is inherently related to θ through the objective function $V_{BO}(s_{t+L}; \theta)$. This relationship can be exploited to infer θ from the data on (a_t, s_t). Hoki used some variant of the discrete-choice regression method to determine the values of θ.[15]

Performance. Bonanza won the world championship in computer shogi in 2006 and 2013. In 2007, the Ryūō ('dragon king', one of the two most prestigious titles) champion Akira Watanabe agreed to play against Bonanza and won. After the game, however, he said he regretted agreeing to play against it, because he felt he could have lost with non-negligible probabilities. Hoki made the source code publicly available. The use of data and machine learning for computer shogi was dubbed the 'Bonanza method' and copied by most of the subsequent generations of shogi programs.

Issei Yamamoto, a programmer, named his software Ponanza out of respect for the predecessor, claiming that his was a lesser copy of Bonanza. From 2014, Ponanza started playing against itself in an attempt to find 'stronger' parameter configurations than those obtained (estimated) from the professional players' data (Yamamoto, 2017). Eventually, Ponanza became the first shogi AI to beat the Meijin ('master', the other most prestigious title) champion in 2017, when Amahiko Satoh lost two straight games.

4.2. *Bonanza is Harold Zurcher*

Bonanza is similar to Deep Blue. Its main component is an approximate terminal-value function, and the 'optimal' action is determined by backward induction on a truncated game tree of self play (equation 4.1). The only difference is the larger number of parameters (50 million), which reflects the complexity of shogi and precludes any hopes for calibration. Hence, Hoki approached the search for θ as a data-analysis problem.

Accordingly, Bonanza is an empirical model of professional shogi players, in the same sense that Rust (1987) is an empirical model of Harold Zurcher, a Madison, Wisconsin, city-bus superintendent. Rust studied Zurcher's record of engine-replacement decisions and estimated his utility function, based on the principle of revealed preference. This comparison is not just a metaphor. The machine-learning algorithm to develop Bonanza is almost exactly the same as the structural econometric method to estimate an empirical model of Harold Zurcher.

Rust's (1987) 'full-solution' estimation method consists of two numerical optimization procedures that are nested. First, the overall problem is to find θ that makes the model's prediction a_t^*

difficult to capture, because the majority of shogi pieces can only move forward, not backward). However, he found that the use of data from amateur players' online games weakened Bonanza's play, and subsequently abandoned this approach (Hoki, 2012).

[15] Tomoyuki Kaneko states he also used some machine-learning methods as early as 2003 for his program named GPS Shogi (Kaneko, 2012). Likewise, Yoshimasa Tsuruoka writes that he started using logit regressions in 2004 for developing his own program, Gekisashi (Tsuruoka, 2012). But shogi programmers seem to agree that Hoki's Bonanza was the first to introduce data-analysis methods for constructing an evaluation function in a wholesale manner.

(as a function of s_t and θ) fit the observed action-state pairs in the data (a_t, s_t). Second, the nested sub-routine takes a particular θ as an input and solves the model to find the corresponding policy function (optimal strategy),

$$a_t^* = \sigma\left(s_t; V_{BO}\left(\cdot; \theta\right)\right). \tag{4.2}$$

The first part is implemented by the maximum-likelihood method (i.e., the fit is evaluated by the proximity between the observed and predicted choice probabilities). The second part is implemented by value-function iteration, that is, by numerically solving a contraction-mapping problem to find a fixed point, which is guaranteed to exist and is unique for a well-behaved single-agent DP problem. This algorithm is called nested fixed point (NFXP) because of this design.

Hoki developed Bonanza in the same manner. The overall problem is to find θ that makes Bonanza predict the human experts' actions in the data (4.2). The nested sub-routine takes θ as an input and numerically searches for the optimal action a_t^* by means of backward induction. The first part is implemented by logit-style regressions (i.e., the maximum-likelihood estimation of the discrete-choice model in which the error term is assumed to be an i.i.d. type-1 extreme value). This specification is the same as Rust's. The second part proceeds on a truncated game tree whose 'leaves' (i.e., terminal values at $t + L$) are given by the approximate value function $V_{BO}(s_{t+L}; \theta)$, and the opponent is assumed to play the same strategy as itself:

$$\sigma_{-i} = \sigma_i. \tag{4.3}$$

Strictly speaking, Bonanza differs from (the empirical model of) Harold Zurcher in two aspects. Bonanza plays a two-player game with a finite horizon, whereas Harold Zurcher solves a single-agent DP with an infinite horizon. Nevertheless, these differences are not as fundamental as one might imagine at first glance, because each of them can be solved for a unique 'optimal' strategy σ^* in the current context.[16] Thus, Bonanza is to Akira Watanabe and his colleagues what Rust (1987) is to Harold Zurcher.

5. GO: ALPHAGO

5.1. Algorithms and Development Procedures

The developers of AIs for chess and shogi had successfully parameterized the state spaces and constructed approximate evaluation functions. Meanwhile, the developers of computer Go struggled to find any reasonable parametric representation of the board.

Go is more complicated than chess and shogi, with $\left|\mathcal{S}_{go}\right| \approx 10^{171} > 10^{71} \approx \left|\mathcal{S}_{shogi}\right|$. Go has only one type of piece, a stone, and the goal is to occupy larger territories than those of the opponent when the board is full of black and white stones (for players 1 and 2, respectively). However, the 19 × 19 board size is much larger, and so is the action space. Practically all open spaces constitute legal moves. The local positions of stones seamlessly interact with the global ones. Even professional players cannot articulate what distinguishes good positions from bad

[16] An alternating-move game with a finite horizon has a unique equilibrium when i.i.d. utility shock is introduced and breaks the tie between multiple discrete alternatives. Igami (2017, 2018) demonstrates how Rust's NFXP naturally extends to such cases with a deterministic order of moves; Igami and Uetake (2019) do the same with a stochastic order of moves.

ones, frequently using phrases that are ambiguous and difficult to codify. The construction of a useful evaluation function was deemed impossible.

Instead, most of the advance since 2006 had been focused on improving game-tree search algorithms (Yoshizoe and Yamashita, 2012; Otsuki, 2017). Even though the board states in the middle of the game are difficult to codify, the terminal states are unambiguous, with either win or loss. Moreover, a 'move' in Go does not involve moving pieces that are already present on the board; it consists of simply dropping a stone on an open space from outside the board. These features of Go make randomized 'play-out' easy. That is, the programmer can run Monte Carlo simulations in which black and white stones are alternately dropped in random places until the board is filled. Repeating this forward simulation many times, one can calculate the probability of winning from any arbitrary state of the board s_t. This basic idea is behind a method called Monte Carlo tree search (MCTS).

Given the large state space, calculating the probability of winning (or its monotonic transformation $V(s_t)$) for all values of s_t remains impractical. However, a computer program can use this approach *in real time* to choose the next move, because it needs to compare only $|\mathcal{A}(s_t)| < 361 = 19 \times 19$ alternative actions and their immediate consequences (i.e., next states) at its turn to move. Forward simulations involve many calculations, but each operation is simple and parallelizable. That is, computing one history of future play does not rely on computing another history. Likewise, simulations that start from a particular state $s_{t+1} = f(s_t, a)$ do not have to wait for the completion of other simulations that start from $s'_{t+1} = f\left(s_t, a'\right)$, where $a \neq a'$. Such computational tasks can be performed simultaneously on parallel-computing infrastructure. If the developer can use many computers during the game, the MCTS-based program can perform sufficiently many numerical operations to find good moves within a short time.

Thus, MCTS was the state of the art in computer Go programming when Demis Hassabis and his team at DeepMind proposed a deep-reinforcement-learning-based AI, AlphaGo. The four components of AlphaGo are (i) a policy network, (ii) RL, (iii) a value network, and (iv) MCTS.

Component 1: Supervised learning (SL) of the policy network. The first component of AlphaGo is a 'policy network', which is a deep neural network (DNN) model to predict a professional player's move a_t as a function of the current state s_t. It is a policy function as in (4.2), with a particular functional form and 4.6 million 'weights' (i.e., parameter vector ψ).[17]

Like Hoki did for Bonanza, the AlphaGo team determined ψ by using the data from an online Go website named Kiseido Go Server (KGS). Specifically, they used the KGS record on 160,000 games played by high-level (6–9 dan) professionals. A game lasts for 200 moves, on average, and eight symmetric transformations (i.e., rotations and flipping) of the board generate nominally different states. Hence, the effective size of the dataset is

$$\begin{aligned} \text{256 million (action-state pairs)} = {} & \text{160, 000 (games)} \times \text{200 (moves/game)} \\ & \times 8 \text{ (symmetric transformations).} \end{aligned}$$

Note that the sample size is still small (negligible) relative to $\left|\mathcal{S}_{go}\right| \approx 10^{171}$.

The basic architecture of AlphaGo's policy network is a standard convolutional neural network (CNN), which is known to perform well in image-recognition tasks, among others. Its final output is the prediction of choice probabilities across all legal moves based on the following logit

[17] I use ψ to denote the parameter vector of policy function, and distinguish it from the parameter vector of the value function θ.

formula:

$$\Pr\left(a_t = j | s_t; \psi\right) = \frac{\exp\left(y_j\left(s_t; \psi\right)\right)}{\sum_{j' \in \mathcal{A}(s_t)} \exp\left(y_{j'}\left(s_t; \psi\right)\right)}, \tag{5.1}$$

where j indexes actions, and $y_j(s_t;\ \psi)$ is the deterministic part of the latent value of choosing action j in state s_t given parameter ψ. Appendix A describes the details of AlphaGo's DNN functional-form specification.

This specification is 'deep' in the sense that the model contains multiple layers, through which $y_j(s_t)$ is calculated. It is termed a 'neural network' because the layers contain many units of simple numerical operations (e.g., convolution and zero-truncation), each of which transmits inputs and outputs in a network-like architecture, with the analogy of computational nodes as biological neurons that transmit electric signals.

The approximate number of parameters in AlphaGo's policy network is

$$4.6 \text{ million (weights)} = (192 \text{ kernels})^2 \times \left(5^2 + 3^2 \times 11 + 1^2\right),$$

where each of the 'kernels' is a 3 × 3 (or 5 × 5) matrix that is designed to indicate the presence or absence of a particular local pattern, in each 3 × 3 (or 5 × 5) part of the board. Note that the number of parameters of AlphaGo's policy function $|\psi|$ is smaller than the 50 million parameters in Bonanza, despite the fact that Go has a larger state space than shogi.

The supervised learning (i.e., estimation) of ψ uses a standard numerical optimization algorithm to maximize the likelihood function that aggregates the optimal choice probabilities implied by the model and the parameter values. That is, AlphaGo's policy function is estimated by the classical maximum-likelihood method. The team did not add any 'regularization' term in the objective function, which is a common practice in machine learning to improve the out-of-sample prediction accuracy at the expense of biased estimates. Nevertheless, the estimated policy function, $\sigma\left(s_t; \hat{\psi}\right)$, could predict 55.7% of the human players' moves outside the sample, and its top-five move predictions contained the actual human choices almost 90% of the time.[18]

Component 2: Reinforcement learning (RL) of the policy network. The ultimate goal of the AlphaGo team was the creation of a strong AI, not the prediction of human play per se (or the unbiased estimate of ψ). The second ingredient of AlphaGo is the process of RL to make a stronger policy function than the estimated one from the previous step, $\sigma\left(s_t; \hat{\psi}\right)$.

In the context of AlphaGo, 'RL' is a numerical search for a 'better' policy function via a relatively simple policy-function iteration.[19] Its task is to find some $\tilde{\psi} \neq \hat{\psi}$ such that the winning probability is higher under strategy $\sigma\left(s_t; \tilde{\psi}\right)$ than under $\sigma\left(s_t; \hat{\psi}\right)$:

$$\Pr_{win}\left(\sigma\left(s_t; \tilde{\psi}\right), \sigma\left(s_t; \hat{\psi}\right)\right) > \Pr_{win}\left(\sigma\left(s_t; \hat{\psi}\right), \sigma\left(s_t; \hat{\psi}\right)\right), \tag{5.2}$$

where $\Pr_{win}\left(\sigma_i, \sigma_{-i}\right)$ is the probability that player i wins with strategy σ_i against the opponent who uses σ_{-i}, in terms of the average performance across many simulated plays of the game.

Because the outcome of the game depends on both σ_i and σ_{-i}, condition (5.2) does not guarantee the superiority of $\sigma\left(s_t; \tilde{\psi}\right)$ in general (i.e., when playing against any strategies other than $\sigma\left(s_t; \hat{\psi}\right)$). The only way to completely address this issue is to solve the game exactly for

[18] By contrast, a simple parametric (logit without a DNN inside) version of the empirical policy function (for the MCTS purposes) achieved only 27% accuracy, which is still remarkable but less impressive than the DNN version's performance.

[19] Sections 6 and 7 explain RL more broadly.

the truly optimal strategy $\sigma^*(s_t)$, but such a solution is computationally impossible. Accordingly, the development team tried to find 'satisficing' $\tilde{\psi}$, by making each candidate policy play against many different policies that are randomly sampled from the previous rounds of iteration (i.e., various perturbed versions of $\hat{\psi}$ in the numerical search process), and by simulating plays from a wide variety of s_t that are also randomly sampled from those in the data (as well as those from perturbed versions of such historical games).

Component 3: SL/RL of value network. The third ingredient of AlphaGo is the evaluation function, $V(s_t; \theta)$, to assess the probability of winning from any s_t . Constructing such an object had been deemed impossible in the computer Go community, but the team managed to construct the value function from the policy function through simulations. Specifically, they proceeded as follows.

(a) Simulate many game plays between the RL policy function, $\sigma\left(s_t; \tilde{\psi}\right)$, and itself.
(b) Pick many (30 million) different states from separate games in the simulation and record their winners, which generates a synthetic dataset of 30 million (*win/loss*, s_t) pairs.
(c) Use this dataset to estimate the value function that predicts $\Pr_{win}$ from any s_t.

In other words, strategy $\sigma\left(s_t; \tilde{\psi}\right)$ implies certain (distribution of) outcomes in the game against itself, and these outcomes become explicit through simulations:

$$\Pr_{win}\left(\sigma\left(s_t; \tilde{\psi}\right), \sigma\left(s_t; \tilde{\psi}\right)\right). \tag{5.3}$$

Once the outcomes become explicit, the only remaining task is to fit some flexible functional form to predict $\Pr_{win}$ as a function of s_t, which is a plain-vanilla regression (supervised learning) task.

The developers prepared another DNN (CNN) with a design that is similar to the policy network: 49 channels, 15 layers, and 192 kernels. The only differences are an additional state variable that reflects the identity of the player (to attribute the win/loss outcome to the correct player), and an additional computational step at the end of the hierarchical architecture that takes all arrays of intermediate results as inputs and returns a scalar ($\Pr_{win}$) as an output.

Let us denote the estimated value network by

$$V\left(s_t; \hat{\theta}, \sigma_i = \sigma_{-i} = \sigma\left(s_t; \tilde{\psi}\right)\right), \tag{5.4}$$

where the expression $\sigma_i = \sigma_{-i} = \sigma\left(s_t; \tilde{\psi}\right)$ clarifies the dependence of $V(\cdot)$ on the use of a specific strategy by both the focal player and the opponent in the simulation step to calculate $\Pr_{win}$. These notations are cumbersome but help us to keep track of the exact nature of the estimated value function.

Component 4: Combining policy and value with MCTS. The fourth component of AlphaGo is MCTS, the stochastic solution method for a large decision tree. When AlphaGo plays actual games, it combines the RL policy $\sigma\left(s_t; \tilde{\psi}\right)$ and the RL value (equation 5.4) within an MCTS algorithm.

Each of these components can be used individually or in any combinations (Silver et al., 2016, Figure 4). The policy function directly proposes the optimal move from any given state. The value function indirectly suggests the optimal move by comparing the winning probabilities across the next states that result from candidate moves. An MCTS can perform a similar state-evaluation task by simulating the outcomes of the candidate moves. They represent more or less the same concept: approximate solutions to an intractable problem that is known to have an exact solution. Nevertheless, positive ‘ensemble effects’ from combining multiple methods are frequently reported, presumably because different types of numerical approximation errors cancel out each other.[20]

5.2. AlphaGo is built on two-step estimation

Component 1: SL policy network is first-stage CCP estimates. Deep Blue and Bonanza embody parametric value functions, whereas AlphaGo’s first key component (SL policy network) is a nonparametric policy function, which is equivalent to the estimation of conditional choice probabilities (CCPs) in Hotz and Miller’s (1993) two-step method.

In case the reader is unfamiliar with the literature on dynamic structural models, I provide a brief summary. The NFXP method (see Section 4.2) requires the solution of a DP problem, which becomes computationally expensive as the size of the state space increases. Moreover, this solution step has to be repeated for each candidate vector of parameter values θ.

Hotz and Miller (1993) proposed an estimation approach to circumvent this problem. To the extent that the actual choices in the data reflect the optimal choice probabilities that are conditional on the observed state in the data, we can estimate the policy function directly from the data. This procedure is the estimation of CCPs in their first stage. The benefit of this approach is that the procedure does not require solving a fully dynamic model. The cost of this approach is that the requirement for data becomes more demanding.

One should avoid imposing parametric assumptions on the first-stage policy function, because a typical goal of empirical analysis is to find the parameters of the value function (and its underlying structural components, e.g., preference and technology), θ, that are implied by observed actions in data. A priori restrictions on the policy function could potentially contradict the solution of the underlying DP. Thus, being nonparametric and preserving flexible functional forms are crucial for an adequate implementation of the Hotz-Miller method. In this sense, the CCP method is demanding of data. It trades the *computational* curse of dimensionality for the *data* curse of dimensionality.

Given this econometric context, the use of DNN seems a sensible choice of functional form. In one of the foundational works for deep learning, the econometrician Halbert White and his co-authors proved that such a multi-layer model with many nodes can approximate any arbitrary functions (Hornik et al., 1989), as long as the network is sufficiently large and deep (i.e., has a sufficient degree of flexibility) to capture complicated patterns.[21] The sole purpose and requirement of Hotz and Miller’s first stage is to capture the actual choice patterns in the data as flexibly as possible.

Component 2: RL policy network is a ‘counterfactual’ with long-lived players. The making of the RL policy network does not involve raw data. Rather, it is a pure numerical search for (a better approximation of) the truly optimal solution of the game.

(a) In the case of the original AlphaGo, RL starts from the top human players’ strategy $\sigma\left(s_t;\hat{\psi}\right)$ as an initial value, and iteratively searches for a ‘stronger’ strategy $\sigma\left(s_t;\tilde{\psi}\right)$.

[20] This ensemble part involves many implementation details that are beyond the scope of this paper.

[21] See also Chen and White (1999). For an overview on deep learning, see Goodfellow, Bengio, and Courville (2016).

(b) In AlphaGo Zero and AlphaZero, RL starts from a purely random play.[22]

Regardless of the choice of the initial value, ‘RL’ in this context is a relatively simple policy-function iteration (or best-response iteration) that has been used in many economic applications, such as Pakes and McGuire’s (1994, 2001) implementation of dynamic oligopoly games.

In the case of the original AlphaGo, the resulting strategy $\sigma\left(s_t;\tilde{\psi}\right)$ could also be interpreted as an outcome of some ‘counterfactual’ experiment in which the top human players (as embodied in $\hat{\psi}$) had long careers, accumulated additional experience, and learned to play better according to particular forms of adaptation. By the same token, the RL for AlphaGo Zero and AlphaZero represents the learning/adaptation trajectory of a first-time player who starts with a purely random strategy.

Component 3: SL/RL value network is second-stage CCS estimates. According to Maddison et al. (2015) and Silver et al. (2016), AlphaGo’s SL/RL value network is the first successful evaluation function for Go. The procedure to obtain this value function is a straightforward application of the Hotz et al. (1994, henceforth HMSS) conditional choice simulation (CCS) estimator, combined with another DNN to approximate the complicated relationship between $\Pr_{win}$ and s_t in the high-dimensional state space.

The context in the SE literature is as follows. Hotz and Miller (1993) proved the existence of a one-to-one mapping between the policy function and the value function, so that the former can be inverted to estimate the latter. This procedure is implemented by means of matrix inversion in the second stage of their original method. However, this procedure requires the inversion of a large matrix, the size of which increases with $|\mathcal{S}|$ and poses computational problems for the actual implementation.

HMSS (1994) proposed an alternative approach to Hotz and Miller’s second step. They suggested running many forward simulations based on the first-stage CCPs. With sufficiently many simulations, the implied value function and its underlying structural parameters can be estimated. The same principle underlies AlphaGo’s success in constructing a useful evaluation function for Go.

Although AlphaGo was developed for the game of Go and therefore a dynamic game, its development goal is a ‘strong’ program to beat human champions (and not about studying the strategic interactions among multiple human players in the data). Consequently, AlphaGo’s connection to the empirical dynamic-game methods seems limited. For example, Bajari et al. (2007, henceforth BBL) extended HMSS (1994) to dynamic games and proposed a moment-inequality-based estimation approach. The development process of game AIs (including AlphaGo) mostly abstracts from strategic interactions.

Component 4: MCTS and ensemble. The actual play of AlphaGo is generated by a complex combination of the estimated/reinforced policy function, the value function, and MCTS.

The MCTS part involves randomly playing out many games. This ‘random’ play is generated from a simple version of the empirical policy function that resembles a standard logit form (i.e., one without the multi-layer architecture of DNN to compute $y_j(s_t)$ as in equation 5.1). Hence, AlphaGo in the actual game is a hybrid of the following:

[22] See Silver et al. (2017, 2018).

(a) the reinforced version of top human players' strategy (as represented in a deep, convolutional logit-like functional form),
(b) their implicit value function (with a similar DNN specification), and
(c) real-time forward simulation based on the estimated 'quick-and-dirty' policy function (in a simple logit form).

AlphaGo Zero and AlphaZero: All-in-one packages. The new version of the program published in 2017, AlphaGo Zero (Silver et al., 2017), does not use any human data (or handcrafted features to represent the board state). Because it has no empirical (human data-related) component, this new AI is not necessarily an obvious subject for econometrics.[23] But one aspect of its architectural design seems intriguing: a single neural network to perform the functions of both the policy network and the value network in the original version. This single network is larger than the previous networks and is combined with MCTS for a purely simulation-based search for the optimal strategy to play Go. The design change seems reasonable from the modelling perspective because the construction of the policy network, the value network, and the MCTS algorithm in the original AlphaGo was redundant. Policy and value are dual objects: one implies the other. Likewise, MCTS is a search for the optimal strategy on its own. The 'ensemble' effect from combining the three components might have conferred an additional performance gain in the original AlphaGo, but it is ultimately a manifestation of approximation errors within individual components.

6. REINFORCEMENT LEARNING AND APPROXIMATE DP

Whereas the previous sections studied the development procedures of the specific game AIs, the rest of this article compares and contrasts 'AI-related methods' and SE more generally. Section 6 explains RL from the economics perspective, and Section 7 discusses its relationship with (dynamic) SE.

6.1. *What is reinforcement learning?*

RL (also known as approximate dynamic programming: ADP) is a collection of methods to solve DP problems *approximately*. The reason we might want to pursue approximate solutions is that exact solutions require complete models of the environment, including transition probabilities of states,[24] and are computationally expensive when many states exist.[25] For example, consider an infinite-horizon utility-maximization problem (with discrete and finite states) by the value-function iteration algorithm on the Bellman equation,

$$V\left(s\right)=\max_{a\in\mathcal{A}(s)}\sum_{s'}p\left(s'|s,a\right)\left(u\left(s,a,s'\right)+\beta V\left(s'\right)\right), \tag{6.1}$$

[23] See paragraphs on RL policy function in the earlier part of this section. Much of the performance gains in AlphaGo Zero seem to stem from the significantly larger size of the neural network architecture (i.e., a more flexible functional form to parameterize the state space). Hence, its superior performance against the original version does not necessarily speak to the costs and benefits of using human data by themselves.

[24] The term 'model' here mainly refers to the transition probabilities of state variables over time. Exact transition functions could be difficult to specify when the environment entails complicated dynamics.

[25] The computational burden of exact solutions (via backward induction and value-function iteration) has two sources: (a) we have to calculate the numerical values of value functions for *all* states, and (b) this calculation requires calculating the expectations based on *all* state transitions.

where $p(s'|s, a)$ is the state-transition probability from s to s' given action a; $u(s, a, s')$ is the utility (of being in state s, taking action a, and transiting to state s'); and $\beta \in (0, 1)$ is the discount factor. Note that the exact solution requires the calculation of $V(s')$ for all s' at each iteration.

The general idea of RL/ADP is to approximate the optimal value function V^* by some $\tilde{V}$, and then derive its corresponding policy function $\tilde{\sigma}$ by collecting the action $\tilde{\sigma}(s)$ for each s that maximizes the approximate expected present value,

$$\tilde{\sigma}(s) = \arg \max_{a \in \mathcal{A}(s)} \sum_{s'} p\left(s'|s, a\right)\left(u\left(s, a, s'\right) + \beta \tilde{V}\left(s'\right)\right). \tag{6.2}$$

Two basic approaches for approximation are stochastic approximation and functional approximation.[26]

Stochastic approximation works as follows. Instead of solving backward, we can simulate many histories of state transitions and take the average of the values across these forward simulations to calculate the expected values. A simple example of this approach in game AI is the use of Monte Carlo simulations (MCTS) in Go-playing programs.

Functional approximation works as follows. Instead of calculating the exact value for each state, we can approximate the true value function by assuming some (semi)parametric functional form in which the number of parameters is smaller than the number of states. Game-AI examples of this approach are linear board-state-evaluation functions in Deep Blue and Bonanza, and DNNs in AlphaGo. Gaussian processes are another example.

In addition to these basic ideas for approximation, RL emphasizes adaptive, incremental, and experience-based procedures, partly because certain applications require real-time implementations, and partly because RL (as a field of AI research) has its origins in psychology and the study of animal behaviours. The *process* of learning to behave optimally is a focus of research in its own right.

Consequently, many RL methods involve iterative procedures to improve value functions and/or policy functions in a piecewise manner. That is, numerical values are updated in each time period (stage) for each specific state that the agent experiences in real data and/or along simulated histories. This step-by-step aspect of the optimization (learning) process works well with parallel computing, where states and their histories are experienced/simulated often without perfect synchronization of all processors. A game-AI example of such ‘asynchronous’ updating is the ‘ensemble’ part of AlphaGo.

Some methods have been formally proved to converge to the truly optimal solutions, but many others have not. In general, stochastic approximation requires that sample histories reflect the true state transitions ‘well’. For example, Q-learning requires that all states are visited infinitely often in forward simulations (see the next subsection). In general, functional approximation requires that parameterization is sufficiently flexible. For example, DNNs can approximate any functions arbitrarily well *if they contain arbitrarily many nodes and parameters* (Hornik et al., 1989; Chen and White, 1999). Obviously, ‘many simulations and many parameters’ could be as computationally costly as ‘calculating values for many states exactly’. Thus, the practical challenge in any approximation method is to find configurations that are computationally feasible and ‘sufficiently close’ to optimal solutions.

[26] For a concise introduction, see Bertsekas (2019, pp. 12–13 and 222–223).

6.2. Q-learning and alternative-specific value functions

One of the most famous classes of RL algorithms is Q-learning, which is based on an alternative representation of the DP problem that uses action-specific value functions,

$$\bar{V}\left(s, a\right) = \sum_{s'} p\left(s'|s, a\right)\left(u\left(s, a, s'\right) + \beta V\left(s'\right)\right), \tag{6.3}$$

which represents the value of being in state *s after* (conditional on) choosing a particular action *a*. Rust (1987) called this expression an 'alternative-specific value function' in his NFXP algorithm to estimate the parameter of *u* by solving the DP exactly;[27] Watkins (1989) used the term 'action value' (and notation *Q*) when he introduced Q-learning to solve the DP approximately. Hence, it is known as a 'Q-factor', 'Q-value', and 'state-action value' in the RL/ADP literature.

One can use these action-value functions to form an expression that is analogous to the Bellman equation,

$$Q\left(s, a\right) = \sum_{s'} p\left(s'|s, a\right)\left(u\left(s, a, s'\right) + \beta \max_{a' \in \mathcal{A}(s)} Q\left(s', a'\right)\right). \tag{6.4}$$

The value-function iteration on this equation converges to the optimal Q^* from any initial value Q_0, thereby providing another version of the exact solution method. The original Q-learning algorithm is a stochastic and asynchronous-updating version of this procedure, where the expected value on the right-hand side is replaced by some simulation. Appendix B explains further details.

6.3. 'Model-free' methods are not model-free

Some of the RL methods are described as 'model-free' in the literature. Casual followers of AI news often misinterpret this phrase to conclude that certain RL methods obviate the need for modelling. However, these methods are not model-free in the way economists would use the term.

Different research communities use the same word differently. The term 'model' in this context refers specifically to the state-transition probabilities, and the term 'model-free' means only that the researcher does not need to have a completely specified set of equations to describe the law of motion. According to Powell's (2011, p. 122) textbook on ADP, which presents these methods from the operations-research perspective, 'In engineering communities, "model" refers to the transition function ... Model-free dynamic programming refers to applications where we do not have an explicit transition function (the physical problem may be quite complex). In such problems we may make a decision but then have to observe the results of the decision from a physical process.'

Hence, 'model-free' methods provide decision rules (learning algorithms) *for the agent* that could eventually lead to approximately optimal behaviours, without requiring the agent to have any knowledge of transition probabilities. This specific use of the term 'model-free' (in engineering communities including RL/ADP) should not be confused with the false notion that *the researcher's use* of RL methods would not involve any modelling or insights based on mathematical models. The 'model-free' RL methods can work without requiring *the agent* to have any knowledge of transition probabilities, because *the researcher* formulates a DP problem and (usually) exploits the recursive structure of Bellman's optimality equation.

[27] I thank John Rust for pointing out this connection. It is also called 'conditional value function' in the SE literature.

In practice, the researcher's modelling tasks for RL methods include the following.

(a) Formulating a dynamic optimization problem (of sequential decision-making) in a recursive form.
(b) Making sure all relevant physical, informational, and belief states are included as state variables.
(c) Making sure state transitions follow a Markov process.
(d) Specifying the agent's choice set as a function of state.
(e) Specifying the timing of actions and state transitions, including the arrival of information and the evolution of beliefs.

'Model-free' RL methods relax the degree to which the researcher has to specify transition probabilities (i.e., parts of (c) and (e) in the above list), but the law of motion is still a model, and all other modelling tasks need to be performed. In economics, defining an agent's optimization problem and its environment is considered part of modelling. Hence, RL methods are as model-based as any other economic analysis using structural models in general and DP in particular.[28]

7. AI AND STRUCTURAL ESTIMATION

7.1. *What is structural estimation?*

At this point, clarifying what structural estimation is might be helpful. Readers who are familiar with the concept may skip this subsection. The following exposition is based on the foundational works in *Statistical Inference in Dynamic Economic Models* (Cowles Commission for Research in Economics, Monograph No. 10), such as Marschak (1950) and Hurwicz (1950a,b).

A *model* $\mathcal{S}$ is (a priori information on) an equation or a system of equations that are mutually consistent.[29] A *structure* S is all properties of the equation(s), including those not known a priori (i.e., the parameter values). If the model is stochastic, the structure also includes the values of the parameters that govern the distribution of random variables. Hence, a model is a class of structures.[30] Based on this broad definition, almost any economic models and regression equations are *structural models*.

A slightly narrower and perhaps more common definition focuses only on the models with equations that describe plausible behaviours of individual economic agents (e.g., buyers, sellers, consumers, and firms).[31] These *behavioural models* usually entail some kind of individual optimization problems (e.g., utility maximization and profit maximization), but what constitutes

[28] See Powell (2011, ch. 15.5) for general advice on modelling. Powell observes critically that 'there is a long history in the optimization community to first present a complete optimization model of a problem, after which different algorithms may be discussed ... The same cannot be said of the approximate dynamic programming community, where a more casual presentation style has evolved ... there appear to be many even in the research community who are unable to write down an objective function properly.' Subsequently, he advises that 'To properly describe a dynamic program ... we need to specify the value function approximation and the updating strategies used to estimate it ... These steps require a proper description of the state variable (by far the most important quantity in a dynamic program), the decision variable, exogenous information and the contribution function. But as with any simulation problems, it is not always clear that we need the transition function in painstaking detail ... [because] when modeling a dynamic system, it is well known that the transition function captures the potentially complex physics of a problem.'

[29] This symbol denotes a set of structures and should not to be confused with the one for state space in earlier sections.

[30] In general, the structural relations or the distribution do not need to have a parametric form.

[31] The need for such *behavioural models* arises from the fact that most economic variables are chosen by these agents, not by the economist. Marschak and Andrews (1944, p. 144) offer a classic example: 'Can the economist measure the

'individual economic agents' and their 'plausible behaviours' depends on the level and scope of the analysis.[32]

Because economics is fundamentally a public-policy discipline, we often study policy interventions at some level of aggregate activities (e.g., a market, an industry, a country, or a planet). Moreover, many economic questions concern changing not only the states of individual agents, but also the rules governing the environment. A *structural change* is a change in any of the structural (i.e., not directly observable) properties of the system.[33] Assessing the effects of such structural changes requires a model that can simulate individual agents' re-optimized behaviours in the new environment.[34]

7.2. *Similarities and differences*

Regarding dynamic structural models, the term 'structural estimation' usually relates to behavioural models with dynamic optimization across multiple time periods (i.e., the usage follows the narrower definition in the above). A structure in this context includes the parameters of the agents' objective functions in each period (e.g., utility, profit, and welfare) as well as those that govern state-transition probabilities and the distribution of unobservable random variables (e.g., taste and cost shocks). Agents' dynamic optimization is usually formulated as DP problems. Hence, most dynamic structural models use the same kind of equations as other DP-related communities use, including Bellman's optimality equation.

Despite these similarities, the main research goals are different between the AI/RL community and the (dynamic) SE literature. The immediate goal of a typical RL project is to achieve 'sufficiently good' performance for a given task. That is, approximately solving a DP problem is the research objective in its own right. By contrast, the immediate goal of SE is to estimate the structural parameters of (dynamically) optimizing agents, usually as a stepping stone for the ultimate goal of designing an optimal public policy that changes the environment.[35]

Nevertheless, commonalities exist between these seemingly different research activities at a deeper level. An example of shared interest is in the immediate task of SE itself. The estimation of the agents' objective functions is called 'inverse RL' (IRL) in the RL literature (Ng and Russell, 2000), which is currently considered to be one of the frontier areas of research according to Sutton and Barto (2018, ch. 17.4).[36]

effect of changing amounts of labor and capital on the firm's output—the 'production function'—in the same way in which the agricultural research worker measures the effect of changing amounts of fertilizers on the plot's yield? He cannot, because the manpower and capital used by each firm is determined by the firm, not by the economist.'

[32] The term 'behavioural' is used in its classical sense to describe agents' actions, which is different from this adjective's more recent use as a synonym for 'boundedly rational' in behavioural economics.

[33] Hurwicz (1950a, p. 267) provides another classic example: 'Let an equation system consist of a supply equation and a demand equation for some commodity. Then the structure is given by the parameters of these two equations ("the structural parameters") and the parameters of the distribution of the disturbances ... Thus structural changes occur if people's tastes shift or industrial productivity increases, provided neither tastes nor productivity are observed directly; should they have been observed, they could simply have been included among the variables of the system.'

[34] The need for such counterfactual simulations arises because experiments are usually too costly or impossible for aggregate economic activities. Koopmans (1947, p. 167) compares an empirical research on business cycles to Brahé and Kepler's works in astrophysics, and then warns that 'an observed regularity not traced to underlying behavior patterns, institutional rules, and laws of production, is therefore an instrument of unknown reliability.'

[35] For example, Igami and Uetake (2019) estimate a structural model of a dynamic oligopoly game with endogenous mergers and innovation for the purpose of assessing the long-term welfare consequences of counterfactual government policies toward mergers.

[36] Another potential bridge between RL and SE (and economics in general) is to consider the ultimate task of SE (i.e., optimizing public policy) as a goal of large-scale approximate DP problems. Obviously, most of the market-, industry-,

7.3. How could RL be useful for SE?

RL could be useful for SE in multiple ways.

First, RL *as a collection of approximate solution methods for DP problems* could be directly useful for solving dynamic structural models with large state spaces.[37] One may consider using stochastic approximation within a full-solution estimation framework. Even when the researcher uses a two-step estimation method to avoid the burden of solving a DP problem, counterfactual simulations do require solving it eventually. Hence, practical solution methods could become an important part of an SE project.[38] Certain functional approximation methods also seem amenable for use in SE.[39]

Second, RL *as a collection of 'approximately optimal learning' models* might be useful as a potential modelling approach. Such models may offer descriptions of plausible human behaviours in certain institutional contexts. Unbounded rationality is a common assumption in dynamic structural models, and this modelling choice may be favoured as a benchmark for several reasons. In principle, however, it is only one of the many possible behavioural/informational assumptions. Hence, having RL models as alternative assumptions would widen the variety of structural models.

Third, many success stories in the engineering application of RL seem to suggest that many real-world problems, either static or dynamic, are amenable to the DP formulation and solution. The use of DP in economics is usually reserved for explicitly intertemporal, multi-period problems. But its core ideas of recursive formulation and iterative solution could be applicable to a broader set of problems.

Finally, an additional implication of RL's commercial successes in areas such as pricing, advertising, logistics, and inventory management is that the behaviours of real-world decision-makers could actually become dynamically optimal. The increased use of automated optimization tools in firms' operations makes 'profit maximization' an increasingly plausible assumption in the modelling of firms' behaviours. The use of RL and other AI-related technologies would seem to make the real world closer to the simplified world of structural models, making reality ever more amenable to economic analysis.

7.4. How could SE be useful for AI?

The need for 'explainable AI' seems likely to persist.[40] However, many attempts to explain AI are ad hoc. Some papers equate explicability with linearity, sparsity, or certain classes of logical operations, but the choice of functional form is only a part of modelling and does

and country-level public policies are too 'large-scale' to permit sufficient experimentation, which is why SE exists in the first place. Nevertheless, certain policy-design problems might be amenable to RL-style incremental optimization.

[37] For example, Brumm and Scheidegger (2017) use adaptive sparse grids; Scheidegger and Bilionis (2019) and Renner and Scheidegger (2018) use Gaussian processes and Bayesian Gaussian mixture models; and Azinović, Gaegauf, and Scheidegger (2019) propose 'deep equilibrium nets' to solve high-dimensional dynamic problems in economics.

[38] How to deal with approximation errors (especially when one is interested in comparing the outcomes between the estimated model and counterfactual simulations) would be an important open question.

[39] Examples of such econometric methods include Chernozhukov et al. (2018), Chernozhukov et al. (2019), and Kaji et al. (2019).

[40] For example, Yoshiharu Habu, the strongest shogi player in recent history, states he does not understand certaion board-evaluation functions of computer shogi programs (Habu and NHK, 2017). The U.S. Department of Defense airs its concern that 'the effectiveness of these systems is limited by the machine's current inability to explain their decisions and actions to human users', which led it to host the Explainable AI (XAI) program aimed at developing 'understandable' and 'trustworthy' machine learning. See https://www.darpa.mil/program/explainable-artificial-intelligence (accessed on October 17, 2017, and February 17, 2020).

not necessarily determine whether a model explains something in any fundamental manner.[41] Others propose the use of 'counterfactuals' to justify the optimality of AI's recommendations (i.e., by demonstrating that other actions are expected to result in suboptimal outcomes). But this 'explanation' is tautological: AI's recommendation is optimal because, according to the same AI, other actions are suboptimal.[42] Likewise, attempts to incorporate users' responses and psychological experiments to find 'explanations' that human subjects can swallow are more about 'persuading naive human users' than about explaining AI.[43]

The proposal that seems the most reasonable is to build an interpretable model in the first place,[44] which is where the guiding principles of research in SE could serve as a benchmark for what a satisfactory explanation (or 'explainable AI') should look like. The purpose and procedures of a typical SE project is threefold: (a) to understand the agents' motives behind their behaviours, (b) to understand the mechanisms through which outcomes are generated by various economic factors, and (c) to evaluate alternative 'rules of the game' to improve aggregate outcomes.[45] The underlying principle is that the explanations in (b) and the optimal solution in (c) should be grounded in the well-defined objects in (a) and that these objects should be empirically tractable.

Conversely, the analogy of AI as SE also tells us what 'explainable AI' should not be about. Efforts to directly explain AI's recommendations are unlikely to be fruitful in the scientific sense, because an (approximately) optimal policy function cannot be fully understood as a stand-alone object in isolation from the underlying optimization problem.[46] The nature and quality of the solution, in turn, depend on the model, the data, and the procedures for estimation and simulation. What are the objective functions, constraints, and laws of motion? Where do data come from? How are simulations performed? These questions must be asked and answered before anyone can

[41] See Caruana et al. (2015), Lakkaraju et al. (2016), and Tan et al. (2018), for example.

[42] See Byrne (2019), for example.

[43] See Lage et al. (2018), Bansal et al. (2019), Rutjes et al. (2019), and Miller (2019), for example.

[44] Rudin (2019) distinguishes explaining black-box models from designing inherently interpretable models. Note that the meaning of 'interpretable models' could be different between the AI and SE literatures.

[45] Most AI applications solve only single-agent problems and do not have a counterpart to (c) in terms of an aggregate object, but the analogy applies to other parts.

[46] IRL seems another AI/RL subfield with obvious connections to (dynamic) SE. Whereas a standard RL problem is to find an (approximately) optimal policy function given the reward function and state-transition probabilities, an IRL problem is to find an implied reward function given transition probabilities and a policy function (or histories of human experts' actions and states in data). The philosophy underlying IRL is that the reward function, rather than the policy or value function, summarizes the task in the most parsimonious and transferrable way (Ng and Russell, 2000). Many economists would agree. At a more operational level, the motivation for IRL is three-fold. First, estimating models of animal and human behaviours is an important part of many scientific inquiries in general. Second, IRL may provide a faster and computationally more efficient path to the ultimate goal of RL to achieve (approximately) optimal performance. The availability of data on human experts' behaviours allows 'learning from demonstration' or 'apprenticeship/imitation learning', as such methods are called in the RL literature. The 'SL policy network' of AlphaGo is an example of such a shortcut. Third, IRL from human behaviour could potentially lead to more 'explainable', 'human-like', and 'socially acceptable' machine behaviours. A recurring issue in both SE and IRL is how to deal with 'suboptimal' behaviours of humans in data. A typical approach in SE would be to extend the model (e.g., the reward function, the choice set, the discount factor, information, and beliefs) to incorporate and rationalize the observed patterns that exhibit some sort of 'bounded rationality' (see Appendix C for an example of such possibilities with respect to human board-game players). By contrast, a typical IRL approach seems to focus on directly modifying the computational implementation of the optimality constraints (e.g., by permitting some margin of errors). For example, Abbeel and Ng (2004) study the case of reward functions that are linear in parameters. We may interpret this line of work in IRL as an attempt to operationalize the use of revealed-preference conditions (inequalities), such as those in BBL (2007), as systematically as possible. The combination of these approaches would expand the toolbox for both communities.

claim to understand the ‘solution’. We should explain the recipe before trying to explain the dish that comes out of it.

8. AI AS STRUCTURAL ESTIMATION

This article carries three messages. First, AI/RL and SE are closely connected by their common use of dynamic optimization models and methods. Second, the success of AI/RL in some areas suggests that certain methods of approximate DP could be useful for research in economics. Third, economists can (and probably should) assess the products and procedures of AI/RL as if they were empirical papers in SE. This analogy helps us to ask the right questions about the modelling choice, data collection, and estimation/simulation methods that go into AI/RL products. Our answers to these questions would then determine whether and how each AI/RL product can be interpreted and explained. AI should be studied as SE.

ACKNOWLEDGEMENTS

The first version of this paper was circulated on October 30, 2017, under a slightly longer title. I thank Susan Athey, John Rust, Simon Scheidegger, and Takuo Sugaya for many detailed suggestions. This paper also benefitted from seminar comments at Riken AIP, Georgetown University, Tokyo University, Osaka University, Harvard University, MIT, Johns Hopkins University, Stanford GSB, and ‘The Third Cambridge Area Economics and Computation Day Conference’ at Microsoft Research New England, ‘The Second Conference on Dynamic Structural Models: Methodologies and Applications of Structural Dynamic Models and Machine Learning’ at University of Copenhagen, and 2018 ‘NBER Economics of Artificial Intelligence Conference’ at University of Toronto, as well as conversations with Xiaohong Chen, Victor Chernozhukov, Philip Haile, Jerry Hausman, Guido Imbens, Greg Lewis, Robert Miller, Yusuke Narita, Aviv Nevo, Whitney Newey, Anton Popov, Jeff Qiu, Elie Tamer, Kosuke Uetake, Hal Varian, and Yosuke Yasuda.

REFERENCES

Abbeel, P. and A. Y. Ng (2004). Apprenticeship learning via inverse reinforcement learning. *Proceedings of the 21st International Conference on Machine Learning*.

Arcidiacono, P. and R. A. Miller (2011). Conditional choice probability estimation of dynamic discrete choice models with unobserved heterogeneity. *Econometrica 79*, 1823–67.

Athey, S. (2017). Beyond prediction: Using big data for policy problems. *Science 355*, 483–85.

Azinović, M., L. Gaegauf and S. Scheidegger (2019). Deep equilibrium nets. Working paper, HEC Lausanne.

Bajari, P., C. L. Benkard and J. Levin (2007). Estimating dynamic models of imperfect competition. *Econometrica 75*, 1331–70.

Bansal, G., B. Nushi, E. Kamar, W. S. Lasecki, D. S. Weld and E. Horvitz (2019). Beyond Accuracy: The Role of Mental Models in Human-AI Team Performance. *The Seventh AAAI Conference on Human Computation and Crowdsourcing (HCOMP-19)*, 2–11.

Belloni, A., V. Chernozhukov and C. Hansen (2014). High-dimensional methods and inference on structural and treatment effects. *Journal of Economic Perspectives 28*, 29–50.

Berry, S. T. and G. Compiani (2019). An instrumental variable approach to dynamic models. Manuscript, Yale University and UC Berkeley.

Bertsekas, D. P. (2019). *Reinforcement Learning and Optimal Control*, Belmont, MA: Athena Scientific.

Bertsekas, D. P. and J. N. Tsitsiklis (1996). *Neuro-Dynamic Programming*, Belmont, MA: Athena Scientific.

Brumm, J. and S. Scheidegger (2017). Using adaptive sparse grids to solve high-dimensional dynamic models. *Econometrica 85*, 1575–612.

Byrne, R. M. J. (2019). Counterfactuals in explainable artificial intelligence (XAI): Evidence from human reasoning. *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI-19)*, 6276–82.

Campbell, M., A. Hoane and F. Hsu (2002). Deep Blue. *Artificial Intelligence 134*, 57–83.

Caruana, R., Y. Lou, J. Gehrke, P. Koch, M. Sturm and N. Elhadad (2015). Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2015)*, 1721–30.

Chen, X. and H. White (1999). Improved rates and asymptotic normality for nonparametric neural network estimators. *IEEE Transactions on Information Theory 45*, 682–91.

Chernozhukov, V., J. C. Escanciano, H. Ichimura, W. K. Newey and J. M. Robins (2018). Locally robust semiparametric estimation. arXiv:1608.00033v2 [math.ST].

Chernozhukov, V., W. Newey and V. Semenova (2019). Inference on weighted average value function in high-dimensional state space. arXiv:1908.09173 [stat.ML].

Goodfellow, I., Y. Bengio and A. Courville (2016). *Deep Learning*, Cambridge, MA: MIT Press.

Habu, Y., NHK. (2017). *Jinkou chinou no kakushin*, Tokyo, Japan: NHK Shuppan (in Japanese).

Hoki, K. (2012). Kazuno bouryoku de ningen ni chousen! Bonanza no tanjou. In Computer Shogi Association (Ed.), *Ningen ni katsu computer shogi no tsukuri kata*. Tokyo, Japan: Gijutsu Hyouron Sha, 135–48 (in Japanese).

Hoki, K. and A. Watanabe. (2007). *Bonanza Vs Shoubunou: Saikyou shogi sohuto wa ningen wo koeruka*, Tokyo, Japan: Kadokawa (in Japanese).

Hornik, K., M. Stinchcombe and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 359–66.

Hotz, V. J. and R. A. Miller (1993). Conditional choice probabilities and the estimation of dynamic models. *Review of Economic Studies 60*, 497–529.

Hotz, V. J., R. A. Miller, S. Sanders and J. Smith (1994). A simulation estimator for dynamic models of discrete choice. *Review of Economic Studies 61*, 265–89.

Hurwicz, L. (1950a). Generalization of the concept of identification. In T. Koopmans (Ed.), *Statistical Inference in Dynamic Economic Models*, Cowles Commission for Research in Economics, Monograph No. 10, London, U.K.: Wiley, 245–57.

Hurwicz, L. (1950b). Prediction and least squares. In T. Koopmans (Ed.), *Statistical Inference in Dynamic Economic Models*, Cowles Commission for Research in Economics, Monograph No. 10, London, U.K.: Wiley, 266–300.

Igami, M. (2017). Estimating the innovator's dilemma: Structural analysis of creative destruction in the hard disk drive industry, 1981–1998. *Journal of Political Economy 125*, 798–847.

Igami, M. (2018). Industry dynamics of offshoring: The case of hard disk drives. *American Economic Journal: Microeconomics 10*, 67–101.

Igami, M. and K. Uetake (2019). Mergers, innovation, and entry-exit dynamics: Consolidation of the hard disk drive industry, 1996–2016. *Review of Economic Studies*, forthcoming.

Iskhakov, F., J. Rust and B. Schjerning (2016). Recursive lexicographical search: Finding all Markov perfect equilibria of finite state directional dynamic games. *Review of Economic Studies 83*, 658–703.

Kaji, T., E. Manresa and G. Pouliot (2019). Artificial intelligence for structural estimation. Presentation slides at UC San Diego.

Kaneko, T. (2012). GPS Shogi no tanjou. In Computer Shogi Association (Ed.), *Ningen ni katsu computer shogi no tsukuri kata*. Tokyo, Japan: Gijutsu Hyouron Sha, 117–33 , (in Japanese).

Kasahara, H. and K. Shimotsu (2009). Nonparametric identification of finite mixture models of dynamic discrete choices. *Econometrica 77*, 135–75.

Kasparov, G. (2007). *How Life Imitates Chess: Making the Right Moves, from the Board to the Boardroom*, London, U.K.: Bloomsbury.

Koopmans, T. C. (1947). Measurement without theory. *Review of Economics and Statistics 29*, 161–72.

Lage, I., A. S. Ross, B. Kim, S. J. Gershman and F. Doshi-Velez (2018). Human-in-the-loop interpretability prior. *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018).*

Lakkaraju, H., S. H. Bach and J. Leskovec (2016). Interpretable decision sets: A joint framework for description and prediction. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016).*

Maddison, C. J., A. Huang, I. Sutskever and D. Silver (2015). Move evaluation in go using deep convolutional neural networks. *International Conference on Learning Representations (ICLR).*

Marschak, J. (1950). Statistical inference in economics: An introduction. In T. Koopmans (Ed.), *Statistical Inference in Dynamic Economic Models*, Cowles Commission for Research in Economics, Monograph No. 10, London, U.K.: Wiley, 1–50.

Marschak, J. and, Jr. W. H. Andrews (1944). Random simultaneous equations and the theory of production. *Econometrica 12*, 143–205.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence 267*, 1–38.

Mullainathan, S. and J. Spiess (2017). Machine learning: An applied econometric approach. *Journal of Economic Perspectives 31*, 87–106.

Ng, A. Y. and S. J. Russell (2000). Algorithms for inverse reinforcement learning. *Proceedings of the 17th International Conference on Machine Learning*.

Norets, A. (2012). Estimation of dynamic discrete choice models using artificial neural network approximations. *Econometric Reviews 31*, 84–106.

Otsuki, T. (2017). *Saikyou igo AI AlphaGo kaitai shinsho*, Tokyo, Japan: Shoeisha (in Japanese).

Pakes, A. and P. McGuire (1994). Computing Markov-perfect Nash equilibria: Numerical implications of a dynamic differentiated product model. *RAND Journal of Economics 25*, 555–89.

Pakes, A. and P. McGuire (2001). Stochastic algorithms, symmetric Markov perfect equilibrium, and the 'curse' of dimensionality. *Econometrica 69*, 1261–81.

Powell, W. B. (2011). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Hoboken, NJ: John Wiley & Sons.

Renner, P. and S. Scheidegger (2018). Machine learning for dynamic incentive problems. Working paper, HEC Lausanne.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence 1*, 206–15.

Rust, J. (1987). Optimal replacement of GMC bus engines: An empirical model of Harold Zurcher. *Econometrica 55*, 999–1033.

Rust, J. (2017). Dynamic programming, numerical. *Wiley StatsRef: Statistics Reference Online*.

Rust, J. (2019). Has dynamic programming improved decision making?. *Annual Review of Economics 11*, 833–58.

Rutjes, H., M. C. Willemsen and W. A. Ijsselsteijn (2019). Considerations on explainable AI and users' mental models. *CHI 2019 Workshop: Where is the Human? Bridging the Gap Between AI and HCI.*

Scheidegger, S. and I. Bilionis (2019). Machine learning for high-dimensional dynamic stochastic economies. *Journal of Computational Science 33*, 68–82.

Schwalbe, U. and P. Walker (2001). Zermelo and the early history of game theory. *Games and Economic Behavior 34*, 123–37.

Silver, D. et al (2016). Mastering the game of Go with deep neural networks and tree search. *Nature 529*, 484–89.

Silver, D. et al (2017). Mastering the game of Go without human knowledge. *Nature 550*, 354–59.

Silver, D. et al (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science 362*, 1140–44.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction (2nd edition)*. Cambridge, MA: MIT Press.

Tan, S., R. Caruana, G. Hooker, P. Koch and A. Gordo (2018). Learning global additive explanations for neural nets using model distillation. *Machine Learning for Health Workshop (ML4H) at the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*.

Tsuruoka, Y. (2012). Gekisashi no tanjou. In Computer Shogi Association (Ed.), *Ningen ni katsu computer shogi no tsukuri kata*. Tokyo, Japan: Gijutsu hyouron sha, 73–93, (in Japanese).

Varian, H. (2014). Big data: New tricks for econometrics. *Journal of Economic Perspectives 28*, 3–28.

Watkins, C. J. C. H. (1989). Learning from delayed rewards. PhD Thesis, University of Cambridge, U.K.

Yamamoto, I. (2017). *Jinkou chinou wa donoyouni shite “Meijin” wo koetanoka?*, Tokyo, Japan: Diamond sha, (in Japanese).

Yoshizoe, K. and H. Yamashita (2012). *Computer Go: Theory and Practice of Monte Carlo Method*, H. Matsubara . Tokyo, Japan:Kyouritsu shuppan, (in Japanese).

Zermelo, E. (1913). Über eine anwendung der mengenlehre auf die theorie des schachspiels. *Proceedings of the 5th Congress of Mathematicians*, E.W. Hobson and A.E.H. Love, Cambridge University Press, Cambridge, U.K., 501–4 (in German).

Co-editor John Rust handled this manuscript.

APPENDIX A: FUNCTIONAL-FORM SPECIFICATION OF ALPHAGO

AlphaGo's policy function uses the following functional form, and its value function has a similar architecture. It consists of 48 input 'channels' (variables), 13 'layers' (stages within a hierarchical architecture), and 192 'kernels' (filters to find local patterns). A complete review of deep neural networks (DNNs) in general (or AlphaGo's model specification in particular) is beyond the scope of this paper, but these objects interact as follows. Each of the 48 channels represents a binary indicator variable that characterizes s_t:

$$x_{kt} = \begin{cases} 1 \text{ if feature } k \text{ is present in } s_t, \text{ and} \\ 0 \text{ otherwise.} \end{cases} \tag{A.1}$$

'Features' include the positions of black stones, white stones, and blanks (see Extended Data Table 2 of Silver et al. (2016) for the full list).

These x_{kt}s are not combined linearly (as in V_{DB} or V_{BO}) but are processed by many kernels across multiple hierarchical layers. In the first layer, each of the 192 kernels is a 5 × 5 grid with 25 parameters that responds to a particular pattern within 25 adjacent locations.[47] As the name 'kernel' suggests, this 5 × 5 matrix is applied to perform convolution operations at every one of the 225 (= 15 × 15) locations within the 19 × 19 board:

$$z_{r,c} = \sum_{l=1}^{192} \sum_{p=1}^{5} \sum_{q=1}^{5} w_{l,p,q} \times x_{l,r+p,c+q} + b, \tag{A.2}$$

where $z_{r,\,c}$ is the result of convolution for row r and column c; $w_{l,\,r,\,q}$ is the weight for kernel l, row r, and column c; (p, q) denote the row and column of the kernel; $x_{l,\,r\,+\,p,\,c\,+\,q}$ is an input; and b is the intercept term ('bias'). The weights and intercepts constitute the parameters of the model (collectively denoted by ψ in the main text). DNNs of this type are called convolutional neural networks (CNNs) in the machine-learning literature and are primarily used for image-recognition tasks. The results of convolution are subsequently transformed by a function,

$$y_{r,c} = \max\left\{0, z_{r,c}\right\}, \tag{A.3}$$

where $y_{r,\,c}$ is the transformed output (to be passed on to the next layer as input). This function is called a rectified linear unit (or 'ReLU'). The resulting 15 × 15 output is smaller than the 19 × 19 board; the margins are filled by zeros to preserve the 19 × 19 dimensionality ('zero padding').

In the second layer, another set of 192 kernels is used to perform convolution on the outputs from the first layer. The results go through the ReLU transformation again, and proceed to the third layer. The size of the kernels in layers 2 to 12 is 3 × 3, instead of 5 × 5 in layer 1. In layer 13, the size of the layer is 1 × 1 because the goal of the policy network is to put a number on each of the 19 × 19 board positions without 'zero padding' the margins. Each of the 19 × 19 outputs in this last layer goes through a logit-style monotonic ('softmax') transformation into the [0, 1] interval,

$$CCP_{r,c} = \frac{\exp\left(y_{r,c}\right)}{\sum_{r'} \sum_{c'} \exp\left(y_{r',c'}\right)}, \tag{A.4}$$

so that the final output can be interpreted as the players' conditional choice probabilities of choosing action j (or board location (r, c)).

APPENDIX B: Q-LEARNING

The original Q-learning algorithm is a stochastic and asynchronous-updating version of the procedure in Section 6.2. Make a synthetic dataset by simulating an infinitely long sequence of state-action pairs $\{(s_k, a_k)\}$ and a successor state s'_k for each pair (s_k, a_k). At each iteration k, the action value of (s_k, a_k) is updated

[47] The 'patterns' that these kernels are designed to pick up should be distinguished from the initial 48 'features' (variables) representing the board state in the original input data.

according to

$$Q_{k+1}(s,a) = \left(1-\gamma^k\right) Q_k(s,a) + \gamma^k Q_k'(s,a), \tag{B.1}$$

for all (s, a), where $\gamma^k \in (0, 1]$ is a step size and

$$Q_k'(s,a) = \begin{cases} u\left(s_k, a_k, s_k'\right) + \beta \max_{a' \in \mathcal{A}(s_k')} Q_k\left(s_k', a'\right) & \text{if } (s,a) = (s_k, a_k), \\ Q_k(s,a) & \text{if } (s,a) \neq (s_k, a_k). \end{cases} \tag{B.2}$$

The convergence to the optimal action-value function requires that (a) all state-action pairs (s, a) are generated infinitely often in the infinitely long sequence of $\{(s_k, a_k)\}$; (b) the successor state s' at each occurrence of a given (s, a) pair is sampled independently; and (c) the step size satisfies the following regularity conditions: $\gamma^k > 0$ for all k, $\sum_{k=0}^{\infty} \gamma^k = \infty$, and $\sum_{k=0}^{\infty} \left(\gamma^k\right)^2 < \infty$. Condition (a) suggests the need for exhaustive exploration, which creates the exploration–exploitation tradeoff in practice.

APPENDIX C: IMPLICIT ASSUMPTIONS OF THE GAME-AI MODELLING

This section discusses some of the implicit assumptions underlying the procedures to develop these AIs, and discusses their implications.

Deep Blue does not use data analysis (at least for its main component), but both Bonanza and AlphaGo use logit-style discrete-choice models and therefore implicitly assume the presence of an error term associated with each of the available actions $a_t \in \mathcal{A}(s_t)$: $\varepsilon(a_t) \sim$ type-1 extreme value.[48] The addition of this continuous random variable eliminates the possibility of ties between the payoffs of multiple actions j and j', thereby making the mapping between the value function and the policy function unique.

In the plain-vanilla application of discrete-choice models, including Bonanza and AlphaGo, this error term is assumed i.i.d. across actions, players, time, and games. In some empirical contexts, however, one might want to consider relaxing this distributional assumption.

(a) Consideration sets and selective search For example, a subset of legal moves $\mathcal{C} \subset \mathcal{A}(s_t)$ could belong to *joseki*, or commonly known move patterns, whereas other moves are not even considered by human experts (i.e., are outside their 'consideration sets') unless new research shows their effectiveness. Similarly, experts are forward-looking but focus on only a few moves per decision node, conducting a highly selective game-tree search. Capturing these aspects of human play would require the econometrician to distinguish between choice sets and consideration sets.

(b) Cross-sectional heterogeneity Large $|\mathcal{A}|$, $|\theta|$, and $|\psi|$ necessitate the data-analysis part of the AIs to pool data from all games regardless of the identity of players or occasions. However, players are heterogeneous in their styles and strengths. Such differences would become a source of systematic heterogeneity in $\varepsilon(a_t)$ across players and contradict the i.i.d. assumption.

(c) Inter-temporal heterogeneity Likewise, the state of knowledge about desirable moves, or *joseki*, evolves over time as a result of new games, experimentation, and research (including the emergence of game AIs and their play styles). The evolving nature of knowledge and strategies becomes a source of systematic heterogeneity in $\varepsilon(a_t)$ across time.

(d) Strategic interactions Each game in the data embodies two specific players' attempts to outmanoeuvre each other. When experts prepare for games, they study rival players' past and recent strategies to form beliefs about their choice probabilities and exploit any weaknesses. Such interactions continue during the actual games, as players keep updating their beliefs and adjust their strategies accordingly. By contrast, both the development process and the actual play of game AIs are based on the assump-

[48] Because the goal of these games is to win, $\varepsilon(a_t)$ does not contribute to the eventual payoff $u_i(s_t)$. The AIs' formulation treats $\varepsilon(a_t)$ as a purely transient, random component of payoffs that players care about only at the time of choosing concurrent a_t.

Table 1. Dialects of DP.

Object	Economics	Optimal control theory	Reinforcement learning
State variable	x, s (continuous, discrete)	x, i (continuous, discrete)	s (discrete)
Choice variable	i, d, a (choice, decision, action)	u (control)	a (action)
Utility/profit function	u, π	g (cost per stage)	r (reward function)
Transition probability	f, p	$f, p_{ij}(u), p(j\|i, u)$ (system function, transition probability)	$p(s^{'}\|s, a), P_{s,a}(s^{'})$
Value function	V	J (cost function)	V
Policy function	P, σ (CCP, strategy)	π, μ	π
Discount factor	β, δ	α	γ
Optimization problem	Maximization	Minimization	Maximization

Note: This table lists selected examples from Bertsekas (2019, p. 45), Powell (2011, p. 16), Sutton and Barto (2018, pp. xix–xxii), and various papers and books in economics

tion that $\sigma_{-i} = \sigma_i$ (and more or less equivalently, $V_{-i} = V_i$), abstracting from any further strategic interactions.

(e) Time and physical constraints The exposition so far has abstracted from the notion of time constraints, but official games impose limits on the amount of time each player can spend on thinking. This time constraint makes the game nonstationary in terms of clock time, and adds another dimension to the player's optimization problem (in terms of the intertemporal allocation of thinking time). The data analysis for the game AIs abstracts from these fundamental aspects of human play, although computers face the same constraint in actual games. For AIs, time constraints manifest themselves either as the length L of a truncated game tree (in the case of Deep Blue and Bonanza) or as the number of play-out simulations for MCTS (in the case of AlphaGo).

Similar constraints exist in terms of physical (or mental) capacity and in terms of computation speed, information storage, and precision of these operations. Human players are more constrained than computers and more prone to obvious mistakes, especially under severe time constraints. In fact, an important aspect of strategic interactions between human experts is about encouraging the opponent to make mistakes. Mistakes would make the implicit $\varepsilon(a_t)$ irregular, and explicitly incorporating time and physical constraints would lead to different modelling approaches. More recent econometric methods to deal with unobserved heterogeneity in dynamic structural models might help, such as in Kasahara and Shimotsu (2009), Arcidiacono and Miller (2011), and Berry and Compiani (2019).

APPENDIX D: DIALECTS OF DP

Different communities use different languages and notations. Table 1 lists some of the most common examples in DP. Powell (2011) encourages students to familiarize themselves with a variety of 'dialects of dynamic programming', because 'As different communities discovered the same concepts and algorithms, they invented their own vocabularies to go with them' (p. 15). At the same time, he points out that 'the cosmetic differences in vocabulary and notation tend to hide more fundamental differences in the nature of the problem being addressed by each community' (p. 16). For example, the action set in RL is typically small and discrete, whereas control u in control theory is a low-dimensional continuous vector. He further notes that in the context of operations research, control is often denoted by x and could have high dimensionality (e.g., hundreds or thousands).